MASTER EN SCIENCES INFORMATIQUES

Représentation, gestion et exploitation de données hétérogènes en e-Health

Hainaut, Jean-Baptiste

Award date:
2011

Faculté Universitaire Notre-Dame de la Paix
Institut d'Informatique
Année Académique 2010-2011

Représentation, gestion et exploitation de données hétérogènes en e-Health

Jean-Baptiste Hainaut

Mémoire présenté en vue de l'obtention du grade de master en sciences informatiques à finalité spécialisée

Faculté d'Informatique - rue Grandgagnage, 21
B-5000 Namur, Belgique
Tél : +32 (0)81 72 50 02 - Fax : +32 (0)81 72 49 67

Table des matières

Résumé

La plupart des Systèmes de Recherche d'Information (SRI) ne prennent pas en compte les dimensions structurelle et sémantique des documents qu'ils indexent. Bien qu'une multitude de travaux existent dans le domaine, ils ne s'intéressent généralement qu'à une de ces dimensions. Pourtant, les combiner permettrait d'améliorer la pertinence du SRI. Dans ce mémoire, nous envisageons la construction d'un SRI exploitant des documents non structurés, des documents semi-structurés et des données structurées, décrits par une ontologie du domaine. L'information structurelle nous permettra de ne renvoyer que les granules documentaires pertinentes. L'ontologie du domaine nous permettra de baser l'indexation des informations contenues dans les documents sur le sens, plutôt que sur la syntaxe.

Dans la première partie, nous verrons comment représenter les descriptions de ces documents hétérogènes dans une base de données commune. La description d'un document comprend un index, représentant son contenu, et un modèle, décrivant sa structure. On s'intéressera aux documents textes comme exemple de documents non structurés, aux documents XML comme exemple de documents semi-structurés et aux bases de données comme exemple de données structurées. Afin de leur ajouter une dimension sémantique, nous verrons comment mettre ces différents types de descriptions en correspondance avec une ontologie du domaine.

Dans la seconde partie, nous abordons des techniques permettant d'exploiter ces descriptions afin de répondre à une requête utilisateur. Ce dernier doit pouvoir interroger les différents types de documents de manière transparente.

Mots clés

Recherche d'Information, Recherche d'Information structurée, Ontologie, XML, Base de Données

Abstract

Most Information Retrieval Systems (IRS) do not take into account the structural and semantic dimensions of the indexed documents. Although a lot of studies exist in the field, they generally concern only one of these aspects. We think that combining both dimensions may improve the relevance of IRS. In this work, we are studying the building of an IRS encompassing unstructured documents, semi-structured documents and structured data, described by ontology. The structural information allows returning only relevant documentary granules. The ontology enables to index the information retrieved from the documents, based on meaning rather than syntax.

In the first part, we analyse how to represent descriptions of heterogeneous documents in a common database. The description of a document includes an index representing the contents and a model describing the structure. We will focus on text documents as model for non-structured documents, XML documents for semi-structured documents and databases for structured data. In order to add a semantic aspect, we will see how to correlate these types of descriptions with ontology of the field.

In the second part, we consider techniques for using these descriptions to answer a user request. Different types of documents have to be questioned seamlessly.

Keywords

Information Retrieval, Structured Information Retrieval, Ontology, XML, DataBase

Avant-propos

Je tiens à remercier toutes les personnes qui ont participé, directement ou indirectement, à la réalisation de ce mémoire :

- Mon promoteur, le Professeur Jean-Luc Hainaut, pour avoir supervisé ce travail.
- Je remercie mon maître de stage Fabrice Estiévenart, de m'avoir guidé et soutenu tout au long de mon stage au CETIC (Centre d'Excellence en Technologies de l'Information et de la Communication).
- Le personnel du CETIC pour m'avoir accueilli durant mon stage.
- Ma famille et mes amis pour m'avoir soutenu moralement pendant toutes ces années d'études.

Introduction

La Recherche d'Information (RI) vise à développer des Systèmes permettant de répondre aux questions que se posent des utilisateurs, en sélectionnant les informations susceptibles d'y répondre, dans un corpus constitué de documents. La quantité de documents disponibles sur support numérique augmentant de manière exponentielle, ces Systèmes de Recherche d'Information (SRI) éprouvent des difficultés à répondre aux attentes des utilisateurs, qui veulent une information de plus en plus ciblée sur leurs besoins.

La plupart des SRI actuels, y compris Google, perçoivent le document comme une entité atomique composée d'un ensemble de mots-clefs. L'utilisateur exprime son besoin en information à l'aide d'une requête contenant les mots qu'il souhaite voir apparaître dans les résultats de sa recherche ; le SRI recherche alors les documents contenant les mêmes termes que ceux utilisés dans la requête. L'évaluation de pertinence d'un document par rapport à la requête se fait donc sur base de critère purement syntaxique. Ainsi l'utilisateur qui recherche des informations sur les GSM peut utiliser le mot-clef « GSM » ; il trouvera les documents contenant ce mot, mais manquera ceux qui contiennent « Téléphone portable », « Cellulaire », « Nokia 8080 », ... Une solution à ce problème est de rajouter une dimension sémantique aux descriptions des documents. Une autre limite des SRI actuels est qu'un document peut traiter d'une multitude de sujets alors qu'un seul intéresse l'utilisateur. Si un document parle de tous les types de téléphones, tout le document sera renvoyé à l'utilisateur alors qu'il est seulement intéressé par la partie traitant des téléphones portables. Pour améliorer la spécificité des résultats renvoyés, il faut prendre en compte la dimension structurelle des documents.

Pour cibler l'information pertinente dans un document, il faut que la machine puisse comprendre comment son auteur a structuré ses idées. Les bases de données et les langages de balisage comme XML permettent de formaliser la structure sémantique d'un document ; on parle alors de structure logique. La structure logique permet de découper un document en unités pertinentes pour un besoin en information spécifique. Pour ce besoin, il est donc judicieux de ne renvoyer que cette unité d'information plutôt que le document entier. La figure ci-dessous présente un exemple de document structuré, présentant différents GSM. La structure n'est pas destinée à être perçue par l'utilisateur, mais est exploitée par des programmes informatiques, notamment pour la mise en page.

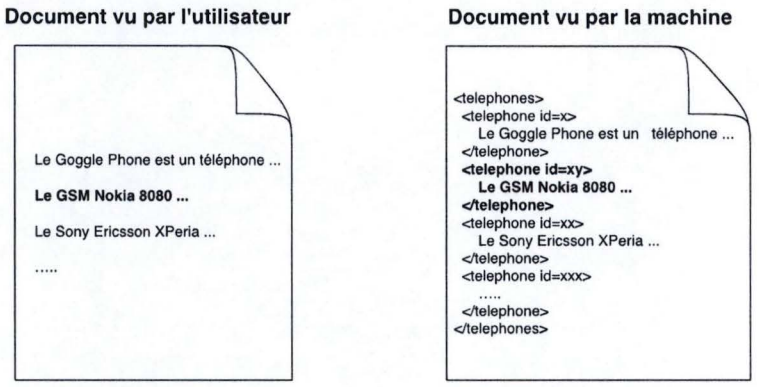


Figure 1 Exemple de document structuré

Du point de vue de la recherche d'information, la structure peut être utilisée pour cibler la partie du document pertinente pour une requête donnée. Pour un utilisateur souhaitant des informations sur le « Nokia 8080 », la partie pertinente, dans l'exemple de document ci-dessus, est celle mise en gras. Avec un SRI traditionnel, le système aurait renvoyé le document entier. L'objectif des SRI structurés est de se servir de la structure logique des documents pour améliorer le niveau de réponse.

Le deuxième aspect qui peut être amélioré dans les SRI est d'ajouter une dimension sémantique à la description des documents. On parle de description pour bien mettre en avant le fait que la représentation que le SRI a d'un document est composée d'informations sur son contenu, l'index, et d'informations sur sa structure. Les ontologies qui formalisent les concepts d'un domaine d'activité ainsi que les relations sémantiques qui les lient permettent d'ajouter cette dimension sémantique. Pour cela, on va mettre en correspondance les concepts d'une ontologie du domaine avec les composants d'une description de document, afin de leur donner du sens.

L'utilisation conjointe d'ontologie du domaine et de la structure logique, par un moteur de recherche, lui permet de répondre plus intelligemment aux requêtes de l'utilisateur.

Ce mémoire s'inscrit dans le cadre du projet « e-Health »[1], une plateforme destinée à permettre l'échange électronique de données entre acteurs de la santé. L'objectif qui sous-tend ce projet est de rendre le dossier du patient accessible aux praticiens traitant le patient, quel que soit l'endroit où ils exercent. Elle permettra, par exemple, à un médecin généraliste, travaillant à domicile, d'avoir accès aux données concernant le séjour à l'hôpital d'un de ses patients, par le réseau internet. Un autre cas d'utilisation est le suivi d'un patient à domicile à l'aide de capteurs. Pour ce projet, il peut être utile de représenter différents types de documents et de disposer d'un Système de Recherche d'Information permettant de les exploiter.

C'est pour cela que nous proposons dans ce mémoire, une méthode pour la représentation des documents non structurés, des documents semi-structurés et des données structurées dans une base de données commune. Nous n'enregistrons pas tout le contenu d'un document, mais uniquement sa description. Celle-ci est constituée d'un index représentant son contenu et, dans le cas où le document dispose d'une structure logique, du modèle définissant cette structure. Nous représentons aussi l'ontologie du domaine dans la base de données. L'utilisation d'une base de données commune permet notamment d'y enregistrer les correspondances entre l'ontologie du domaine et les descriptions auxquelles elle donne du sens, mais aussi toutes correspondances pouvant exister entre descriptions de documents.

Nous verrons ensuite comment exploiter ces données, pour répondre à une requête utilisateur, afin de retrouver l'information pertinente contenue dans les documents. Nous utilisons l'ontologie du domaine pour mettre la requête en relation avec les composants des descriptions de documents pertinents. Une fois ces composants identifiés, nous pouvons dire où se trouve l'information dans le document original. Nous verrons ensuite comment interroger le système contenant l'information pertinente. Ce traitement est différent selon qu'il s'agit d'un document « plat », d'un document semi-structuré ou d'une base de données (structurée).

Ce mémoire est divisé en cinq chapitres :

- Le premier chapitre, intitulé « Etat de l'art » fait état des principes fondamentaux des disciplines en relation avec nos travaux. La première section détaille les différentes étapes du processus de Recherche d'Information ainsi que les principaux modèles qui concrétisent le processus de RI. La deuxième section fait un tour d'horizon des techniques utilisées pour tenir compte de la dimension structurelle des documents. La dernière section du chapitre présente les ontologies comme langage de représentation de connaissance. On verra ce qu'elles apportent à la RI.

[1] eHealth : https://www.ehealth.fgov.be/fr. Date : 24/08/2011.

- Le deuxième chapitre, intitulé « Modélisation des données hétérogènes », présente notre méthode de représentation des différents types de documents, des ontologies et des relations qui existent entre eux.
- Le troisième chapitre, intitulé « Interrogation des données hétérogènes », aborde le problème de l'exploitation des descriptions modélisées au chapitre deux pour la construction d'un Système de Recherche d'Information exploitant des données hétérogènes.
- Le quatrième chapitre, intitulé « Implémentation d'une interface pour la métabase », parle du développement d'une interface Web permettant d'appliquer les méthodes développées au Chapitre 2.

Chapitre 1

Etat de l'art

1.1. La recherche d'information

1.1.1. Introduction

La *recherche d'information* (RI) est la branche de l'informatique qui étudie la manière de répondre au besoin documentaire d'un utilisateur en cherchant et en sélectionnant des informations pertinentes dans un corpus. La RI s'intéresse aussi à la façon de récolter, représenter et stocker les documents constituant le corpus. L'utilisateur réalise sa recherche à l'aide d'un système de recherche d'informations (SRI). Il exprime généralement son besoin en information auprès du SRI sous la forme d'une requête ; ce dernier lui renverra un ensemble d'informations issues de son corpus, qu'il juge pertinentes par rapport à la requête. Le principal objectif d'un SRI est que les documents jugés pertinents par le système le soient aussi pour l'utilisateur.

Les tâches réalisées par le SRI sont :

- L'indexation des documents, qui permet au système de se créer sa propre représentation des documents pour pouvoir répondre plus efficacement à l'utilisateur ;
- L'appariement requête/document, qui consiste à mettre en relation la requête avec le document ;
- L'extension de la requête est une tâche facultative visant à aider l'utilisateur dans la formulation de son besoin en information.

Dans ce chapitre, nous allons nous intéresser au processus de recherche d'informations traditionnel. En RI traditionnelle, on ne s'intéresse qu'au contenu des documents et on ne tient pas compte d'une éventuelle structure logique organisant ce contenu ; cette problématique est l'objet du chapitre 1.2. Nous allons commencer par passer en revue les différentes tâches d'un SRI. Ensuite, nous passerons en revue les principaux modèles de recherche d'information qui concrétise ses tâches.

1.1.2. L'indexation des documents

L'indexation est le processus lors duquel le système analyse chaque document du corpus afin de repérer les mots les plus significatifs, qui constitueront l'index

terminologique. Cette étape a pour but d'accélérer les temps de traitement des requêtes. Ainsi seuls les termes les plus représentatifs d'un document sont retenus par le système afin d'en décrire le contenu. Le processus d'indexation peut être manuel, semi-automatique ou automatique. Les techniques d'indexation manuelle et semi-automatique ont pour point commun de nécessiter l'intervention d'un opérateur humain. Bien que l'intervention d'un opérateur humain permette d'obtenir une indexation, et donc des réponses, de meilleure qualité, elle coûte cher et est difficilement applicable à des corpus de grande taille (ex. le web). On préférera donc utiliser des techniques d'indexation automatique qui, à quelques exceptions près, sont entièrement réalisées de manière algorithmique. Les principales approches d'indexation sont exposées dans (Anderson & Pérez-Carballo, 2001).

L'indexation automatique comprend différentes étapes, détaillées dans (Frakes, 1992). Il s'agit de :

- l'analyse lexicale
- l'élimination des mots fonctionnels
- la lemmatisation
- la pondération des termes

La première étape, l'analyse lexicale, consiste à transformer la suite de caractère que constitue le contenu du document en une suite de termes. Un terme peut être un mot ou une expression comme « State-of-the-art », « fundp.be » ou encore « B52 ». Pour cela, on définit des règles qui permettront à l'analyseur de distinguer les termes du reste (espaces, chiffres, symboles de ponctuations, ...) et donc de savoir quelles suites de caractères il doit garder, séparer...

Vient ensuite l'étape d'élimination des mots fonctionnels. Ces mots, aussi appelés mots vides, sont ceux qui n'apportent aucune information sur le contenu document. Ce sont des mots comme le, un, dans, ... Plus généralement, les mots vides sont tous ceux qui se retrouvent dans la majorité des documents et donc n'apportent aucune information sur le contenu d'un document particulier. L'élimination des mots fonctionnels est généralement réalisée lors de l'analyse lexicale ; on utilise un anti-dictionnaire qui contient la liste des mots qu'on ne souhaite pas garder dans l'index.

La lemmatisation vise à retrouver la racine des mots. En effet, un mot peut se retrouver sous différentes formes tout en gardant le même sens. Ainsi, les mots « recherche », « recherches », « chercher », « cherchant », ... peuvent être ramené à la même racine. Toutes ses variations seront indexées sous un même terme. Cela permet d'augmenter le nombre de documents pertinents renvoyés par le système. Seulement, cela fait aussi augmenter le nombre de documents non pertinents renvoyés. On trouve une illustration de ce problème dans (Singhal, 2001), où une recherche portant sur la recherche d'information (information retrieval en anglais) renvois un article intitulé

« Information on Golden Retrievers » à cause de la lemmatisation. Dans cet article, on apprend aussi que la lemmatisation est plus ou moins utile selon la langue et qu'une mauvaise lemmatisation peut mener à une dégradation des performances du SRI. Une étude détaillée de ce problème et des solutions est présentée dans (Fagan, 1989).

Jusqu'ici, tous les termes sélectionnés pour constituer l'index sont sur le même pied d'égalité. La dernière étape, la pondération des termes, consiste à leurs assigner un poids pour refléter leur importance dans le document, et plus généralement dans la collection, par rapport aux autres termes. Le poids d'un terme est généralement calculé sur base d'une mesure statistique ; la plus simple étant la fréquence d'apparition du terme dans le document. Les mesures couramment utilisées prennent aussi en compte la fréquence du terme dans le reste de la collection. Elles se basent sur la loi de Zipf. Son auteur, George Kingsley Zipf, a observé (Zipf, 1972) que la fréquence d'utilisation d'un mot est inversement proportionnelle à son rang :

$f(n) = \frac{K}{n}$ où K est une constante et n le rang du mot.

Ainsi, si le mot le plus fréquent revient 600 fois dans le texte (K=600 ; rang =1), le deuxième mot le plus fréquent (rang = 2) apparaitra 300 fois, le troisième (rang = 3) 150 et ainsi de suite. Cette relation entre la fréquence et les rangs des termes dans un document peut aussi être utilisée pour choisir ceux qui seront retenus pour constituer l'index de ceux qui doivent être rejetés, car pas assez représentatifs du contenu du document ; ainsi, on élimine les termes les plus fréquents, qui ne sont pas représentatifs, car il s'agit surtout de mots vides qui se retrouvent dans la plupart des documents du corpus ; mais aussi les termes les moins fréquents qui ne permettent pas de mettre en avant les sujets principaux abordés dans le document.

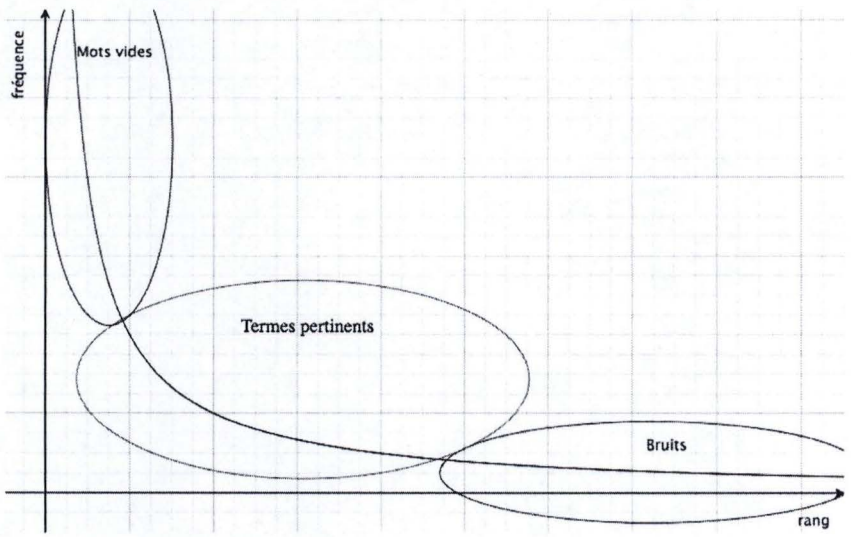


Figure 2 Zone de distribution des termes

TF-IDF est une méthode de pondération très répandue et basée sur la loi de Zipf. Trois facteurs sont combinés pour déterminer le poids d'un terme. La première, tf (Term Frequency), est tout simplement la fréquence du terme dans le document. Cela permet de mesurer l'importance du terme dans le document. La seconde, idf (Inverse Document Frequency), est l'importance du terme par rapport au reste de la collection. Il favorise les termes qui se retrouvent dans peu de documents (et sont donc représentatif du contenu de ces documents) pour défavoriser ceux qui se retrouvent dans tous les documents. Un dernier facteur pouvant être pris en compte est la longueur du document. En effet, un terme présent dans un document plus long que la moyenne aura tendance à avoir un meilleur score, car ces documents utilisent plus de mots et surtout plus de répétition d'un même mot ; il faut donc corriger ce biais induit par la taille du document.

tf est la fréquence du terme dans le document

N est le nombre de documents dans la collection

df_t est le nombre de documents dans la collection qui contiennent le terme t

$$idf_t = log\frac{N}{df_t}$$

$$\text{tf-idf}_{d,t} = tf_{t,d} * idf_t$$

Figure 3 Formules pour le calcul du poids selon tf-idf

Le résultat de l'indexation est finalement stocké en mémoire. La structure la plus efficace est la liste inversée ; pour chaque terme, on référence la liste des documents le référençant. Cette liste prend la forme :

$$t_i \text{ -> } <d_1, \ldots>, \ldots, <d_n, \ldots>$$

où t_i est un des termes formant le vocabulaire du corpus ; d_1, ..., d_n la liste des documents le contenant ; et $<d_n, \ldots>$ une structure de donnée contenant des informations sur le terme dans le document d_n comme les endroits dans le texte où on le retrouve, son poids, ...

1.1.3. La mise en relation requête/document

L'utilisateur exprime son besoin en information à l'aide d'une requête. Avec les premiers systèmes, celle-ci était constituée de mots clefs séparés par des opérateurs booléens (ET, OU, PAS). Seulement, en plus de rendre l'utilisation des systèmes de

recherche difficile d'accès à un utilisateur novice, ces systèmes ne proposaient pas de mesure pour le classement des documents. Les seuls classements possibles étaient basés sur des informations techniques comme la date de création, la taille, ... Si les systèmes booléens sont toujours utilisés, notamment pour l'interrogation de base de données, par des utilisateurs avancés qui les jugent plus efficaces ; les systèmes de recherche d'informations utilisent des mesures qui permettent de classer les documents en fonction de leurs degrés de pertinences par rapport à la requête. Les requêtes sont alors composées d'une liste de mots clefs séparés par des espaces (ex. : 'recherche d'informations structurée').

Concrètement, la requête est vue comme un document et donc indexée selon les mêmes méthodes. Répondre à une requête revient alors à calculer un score de similarité entre deux documents :

> s(Q, D)
> où Q est la requête (mais pourrais être un document du corpus) et D le document.

Ce score sera ensuite utilisé pour classer les documents renvoyés à l'utilisateur par ordre de pertinence. La définition de la fonction s(Q, D) dépend du modèle utilisé pour implémenter la fonction de recherche (voir chapitre 1.1.6). L'avantage de cette approche est de rendre le processus plus intuitif et d'avoir un classement plus naturel des résultats. Cependant, ce type de langage nécessite une courte phase d'apprentissage de la part de l'utilisateur. En effet, il ne vient pas naturellement à l'esprit d'un novice qui se pose la question : « Quelle est la capitale de l'Australie ? » ; d'exprimer sa question en devinant les termes qui apparaitront dans le document ('capitale Australie').

On peut aussi citer l'approche graphique, plus rarement utilisée, où l'utilisateur exprime sa requête en sélectionnant un sujet dans une hiérarchie.

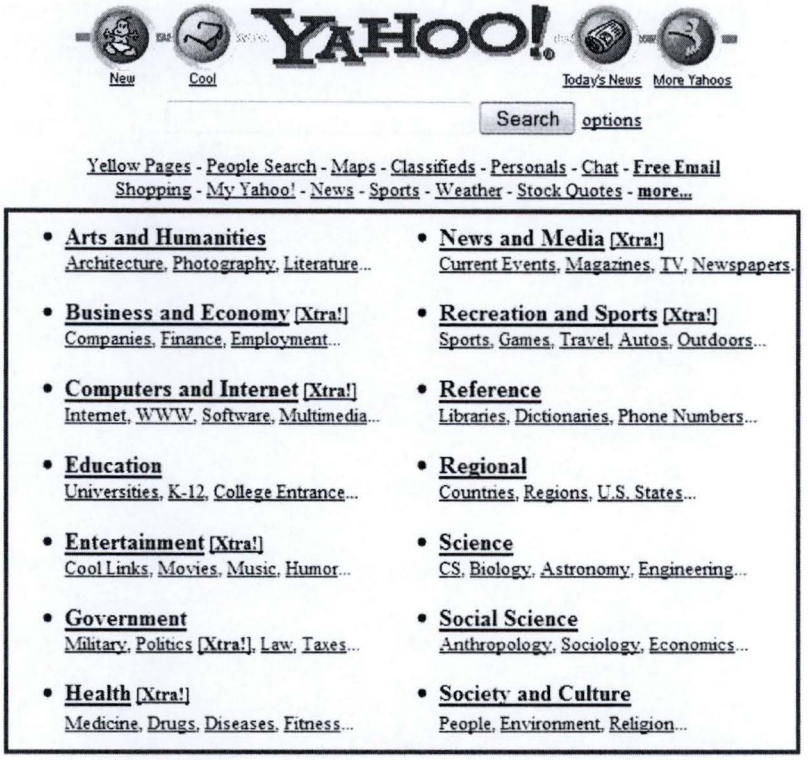


Figure 4 Ancienne version de Yahoo (1998) où l'utilisateur choisi un sujet en naviguant dans une hiérarchie de sujets.

Cette approche présente l'avantage de guider l'utilisateur qui n'a pas d'idée précise sur les termes à employer pour formuler sa requête. L'inconvénient principal survient lorsque l'utilisateur recherche une information couvrant plusieurs sujets, repartis dans différentes catégories.

1.1.4. La reformulation de la requête

La reformulation de la requête, aussi appelée enrichissement, vise à aider l'utilisateur à formuler son besoin en information. En effet, ce dernier, surtout lorsqu'il manque d'expérience, peut avoir du mal à trouver les termes qui lui permettront de trouver des documents pertinents. L'existence de plusieurs dénotations pour qualifier un même sujet rend la tâche de l'utilisateur encore plus ardue. Pour résoudre ce problème, des techniques de reformulation manuelles et automatiques ont été développées. L'idée est de se servir des termes de la requête pour proposer de nouveaux termes : synonymes ou sémantiquement proches. Par exemple pour le mot « avion », les documents contenant son synonyme « aéroplane » seront aussi pertinents ; les termes « aéroport », « bombardier », « jet » sont sémantiquement proche et peuvent aider l'utilisateur à préciser sa requête.

Une première approche consiste à utiliser un thésaurus. Un thésaurus est un dictionnaire qui formalise les relations entre les termes. La constitution manuelle d'un thésaurus prenant beaucoup de temps, des techniques ont été développées pour automatiser ce processus (Jones, 1971). Ces techniques opèrent une analyse statistique des cooccurrences entre les mots du corpus, afin de construire une liste des mots qui se retrouvent souvent ensemble. C'est par exemple le cas du mot « recherche » et du mot « emploi ». Aujourd'hui, des thésaurus « prêts à l'emploi » et portant sur différents domaines sont disponibles sur Internet. On peut citer le cas de WordNet[2] qui formalise des relations lexicales comme la synonymie, l'hyponymie, la métonymie, ... Wordnet est souvent utilisé comme thésaurus pour l'enrichissement, (Voorhees, 1994) mais ne contient malheureusement que des termes de la langue anglaise.

L'approche la plus populaire est la réinjection de la pertinence ou « relevance feedback » (Rocchio, 1971). Cette technique part du principe que l'utilisateur est le seul à pouvoir juger de la pertinence d'un document. Le système commence par afficher une liste de documents en réponse à sa requête. Celui-ci peut alors indiquer ceux qu'il juge pertinents. Le système va alors renforcer le poids de ces documents par rapport aux termes de la requête. Lors d'une prochaine recherche portant sur ces mêmes termes, les documents notés comme pertinents seront mieux classés. Une autre technique, appelée « pseudo-feedback » automatise ce processus (Buckley, Allan, Salton, & Singhal, 1994). Elle consiste à prendre les meilleurs documents renvoyés par le système ; sélectionner des termes fortement liés à ceux de la requête initiale ; et finalement ajouté ces termes à la requête initiale. Cette technique donne de très bons résultats sur des petites requêtes. D'autres systèmes utilisent des informations implicites comme celles contenues dans les historiques de recherches (Cui, Wen, Nie, & Ma, 2002). Ces derniers contiennent, pour chaque requête, les documents choisis par l'utilisateur ; si l'on part de l'hypothèse que l'utilisateur choisit principalement des documents pertinents, on peut se servir de ses informations pour la réinjection de la pertinence. À la différence des systèmes où la notification de la pertinence par l'utilisateur était explicite (Rocchio, 1971), ici elle n'indique qu'un certain degré de pertinence par rapport à la requête. Cette approche est notamment utilisée par Google pour ses suggestions de recherches associées.

[2] http://wordnet.princeton.edu

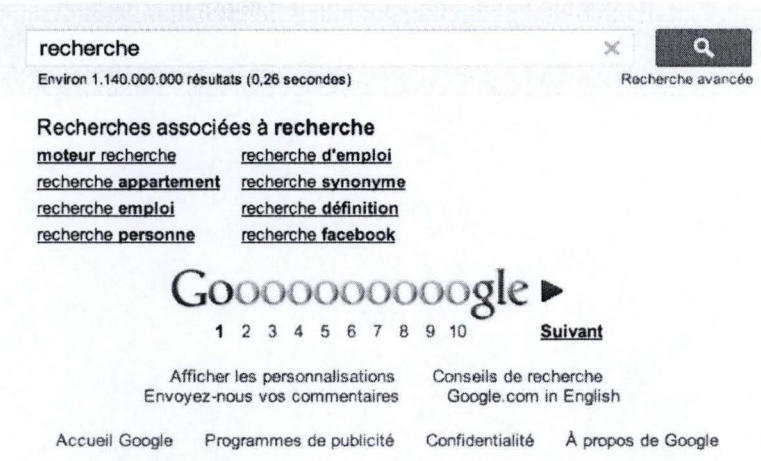


Figure 5 Affichage des termes associés dans les précédentes recherches

1.1.5. Evaluation des systèmes de Recherche d'Information

Un bon système de recherche d'informations doit trouver tous les documents pertinents et rejeter les documents non pertinents. La précision et le rappel sont des mesures permettant d'évaluer un système par rapport à ces objectifs. La précision mesure la proportion de documents pertinents par rapport à l'ensemble des documents retournés par le système. Le rappel mesure la proportion de documents pertinents retournés par le système par rapport à l'ensemble des documents pertinents contenus dans le corpus.

1.1.6. Les différents modèles

Un modèle de RI formalise un processus de recherche d'informations. Les modèles les plus populaires appartiennent à la famille des modèles vectoriels et à la famille des modèles probabilistes.

Modèles vectoriels

Le modèle vectoriel, introduit dans (Salton, Wong, & Yang, 1975), représente les documents dans un espace vectoriel composé d'autant de dimensions qu'il y a de termes dans l'ensemble du corpus. Chaque document est identifié par un ou plusieurs termes indexés auxquels on attribue un poids en fonction de leurs importances. Le vecteur représentant un document i est défini de la façon suivante :

$$D_i = (w_{i1}, w_{i2}, \dots, w_{it})$$

où w_{ij} représente le poids du j^{ème} dans le document i ; t est égal à la taille du vocabulaire du corpus.

Le poids w_{ij} est non nul seulement si le terme j est présent dans le document i. Cette représentation permet de comparer le degré de similarité entre deux documents en

fonction de la distance qui les sépare dans l'espace vectoriel. La requête étant elle aussi considérée comme un document, on peut définir une fonction s(q, d_i) qui mesure le degré similarité entre une requête q et un document d_i. Généralement, cette fonction est définie comme étant le cosinus de l'angle formé par le vecteur q représentant la requête et le vecteur d_i représentant le document i ; Plus formellement :

$$s(q, d_i) = \frac{q * d_i}{\|q\| \, \|d_i\|} = \frac{\sum_{j=1}^{t} w_{qj} * w_{ij}}{\sqrt{\sum_{j=1}^{t} (w_{qj})^2} \sqrt{\sum_{j=1}^{t} (w_{ij})^2}}$$

Cette mesure favorise les termes présents à la fois dans le document et dans la requête ; défavorise les termes qui ne sont présents que dans un des deux vecteurs ; et est neutre pour les termes qui ne sont présents dans aucun des deux vecteurs.

Le modèle ne donne pas d'indications sur le calcul du poids, mais les pondérations selon TF et ITF sont les plus courantes.

Ce modèle classe les documents selon leur pertinence, ce qui permet de renvoyer des documents qui ne satisfont que partiellement la requête. Le langage de requête est proche du langage naturel, car il n'est plus nécessaire de séparer chaque terme par un opérateur booléen.

Le principal inconvénient de ce modèle est qu'il considère que les termes de l'index sont tous indépendants.

Modèles probabilistes

L'idée derrière cette famille de modèles est que le classement des documents d'un corpus doit se faire selon leur probabilité de pertinence à une requête : « le principe de classement probabiliste » énoncé dans (Robertson, 1997). Cette idée fait suite aux recherches de Maron & Kuhns qui définissaient la pertinence d'un document par rapport à un terme indexé comme la probabilité qu'un utilisateur qui utilise ce terme soit satisfait par le document (Maron & Kuhns, 1960). L'aspect le plus intéressant de ce modèle est que la recherche est vue comme un processus itératif, pouvant être guidé par l'utilisateur.

La formule de base développée dans (Robertson, 1997) , classait les documents selon $log \frac{P(R|D)}{P(\bar{R}|D)}$; où P(R|D) est la probabilité que le document D fasse partie de l'ensemble des documents pertinents R ; et $P(\bar{R}|D)$ est la probabilité que le document D fasse partie de l'ensemble des documents non-pertinents $\bar{R}$.

En supposant que la probabilité de pertinence soit indépendante du document considéré, la similarité entre la requête et le document est calculée comme :

$$s(Q,D) = log\frac{P(D|R)}{P(D|\bar{R})}$$

Même si les mesures permettant d'estimer la similarité entre la requête et le document, P(D|R), divergent en fonction du modèle, elle repose sur la même hypothèse. Celle-ci stipule qu'étant donné une requête, il existerait un ensemble « idéal » de documents pertinents. Pour répondre à une requête, le SRI doit pouvoir décrire cet ensemble idéal en spécifiant ces propriétés. Comme le système ne les connaît pas à priori, il se base sur une estimation des propriétés afin de fournir un premier ensemble à l'utilisateur. L'utilisateur désigne les documents pertinents, ce qui permet au système d'améliorer sa description de l'ensemble idéal. Le système continue ainsi pour se rapprocher de cet ensemble.

Cette approche est intéressante, car elle introduit l'idée d'interaction entre le système de recherche d'informations et l'utilisateur. Cependant, elle est très peu utilisée du fait de sa complexité.

Le lecteur intéressé par les modèles probabilistes, trouvera des formules de calcul de la similarité entre une requête et un document dans (Robertson, 1997) et (Singhal, 2001).

1.2. La recherche d'information structurée

1.2.1. Introduction

Les techniques de base de la Recherche d'Information ont été posées lorsque les documents plats étaient majoritaires. Aujourd'hui, les documents structurés et semi-structurés se généralisent avec des formats comme XML. Bien que les systèmes de recherche d'informations classiques fonctionnent avec de tels documents, ils n'exploitent pas leurs structures et les traitent comme de simples documents plats. Or la prise en compte de la structure permettrait d'affiner le niveau de réponse des systèmes qui pourraient ne renvoyer que les granules documentaires pertinentes par rapport au besoin utilisateur.

Pour cela, un SRI peut s'appuyer sur des informations formelles décrivant la structure du document. Une solution est d'encadrer des portions de textes par des balises que des langages comme XML permettent de définir.

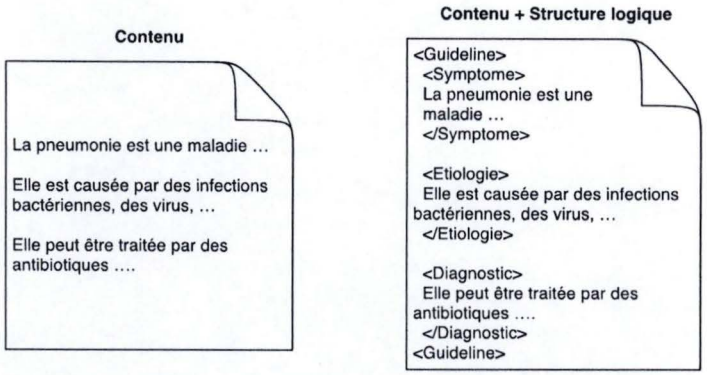


Figure 6 Contenu et structure d'un document

La figure ci-dessus représente le même document ; à droite tel qu'il devra être présenté l'utilisateur ; et à gauche avec des balises rendant explicite la structure sémantique de ce document. En effet lorsqu'un humain lit le document de gauche, il comprend que le premier paragraphe décrit un symptôme, le second son étiologie (les causes de la maladie) et le troisième parle des traitements. L'utilisation de balises, dans le document de droite, pour encadrer ses unités de sens permet de formaliser cette structure sémantique qui pourra être exploitée par un programme informatique.

La structure d'un document XML peut être définie dans une DTD (Document Type Definition) ou un XML Schema[3]. Ces langages de définitions permettent de spécifier le format d'une famille de documents ; pour cela, on définit la forme des balises que ses documents pourront utiliser ainsi que la façon de les composer. Si le format d'un document XML est défini dans une DTD et qu'il respecte les règles qui y sont définies, on dira que ce document est valide. La figure ci-dessous présente un exemple de DTD par rapport à laquelle le document XML décrit à la figure Figure 6 est valide.

```
<?xml version="1.1">!
<!DOCTYPE guideline
"http://www.medical.com/guideline.dtd">!
<!ELEMENT guideline (symptome, ethiologie, diagnostic)>!
<!ELEMENT symptome (#PCDATA)>!
<!ELEMENT etiologie (#PCDATA)>!
<!ELEMENT diagnostic(#PCDATA)>!
```

Figure 7 Exemple DTD

[3] XML Schema : http://www.w3.org/XML/Schema. Date : 27/07/2010.

On remarque qu'une DTD (ou un XML Schema) nous donne des informations sur la façon dont sont structurées les informations contenues dans ses instances.

Prendre en compte la structure du document lors de l'indexation permet de répondre plus précisément au besoin en information de l'utilisateur. Ainsi plutôt que de renvoyer la totalité d'un document qui aborde souvent plusieurs sujets, dont certains n'intéressent pas l'utilisateur ; on va tenter d'augmenter le niveau de granularité pour ne renvoyer que la partie pertinente du document. L'unité d'information n'est plus le document tout entier, mais une de ses composantes. Si un utilisateur interroge un SRI sur les causes de la pneumonie, un SRI classique renverra tout les documents parlant de cette maladie ; en utilisant les informations fournies par le balisage, il est possible de ne renvoyer que les informations contenues dans le nœud « Etiologie » du document représenté à la Figure 6. Ainsi l'unité d'information n'est plus le document entier, mais un élément du document, appelé doxel, pour « Document Element ».

Un SRI performant, renverra l'unité d'information la plus spécifique, mais qui reste exhaustive par rapport au besoin en information (Lalmas, 1997). Une unité d'information est spécifique par rapport à une requête si toutes les informations qu'elle contient concernent la requête. Une unité d'information est exhaustive par rapport à une requête si elle contient toutes les informations pertinentes.

Deux communautés travaillent au problème de la RI structurées : la communauté des bases de données et la communauté RI. La première utilise des techniques et des langages initialement développés pour les Bases de données. Ces techniques nécessitent une connaissance de la structure des documents du corpus. La communauté RI adapte les techniques issues de la RI traditionnelles afin d'exploiter la structure sans que l'utilisateur n'en soit conscient. On s'intéressa surtout aux approches issues de ce courant dit « orientées document » ; bien que beaucoup développent des approches hybrides, utilisant des techniques issues des deux communautés.

Dans ce chapitre, nous allons d'abord voir comment les différentes étapes du processus de RI peuvent être adapté pour prendre en compte la structure logique des documents. Les approches présentées se concentrent toutes sur la RI dans des documents XML.

1.2.2. L'indexation

Comme on l'a vu dans le chapitre 1.1.2 pour les documents « plat », l'indexation consiste à extraire les termes représentant le mieux le contenu document et les pondérer pour refléter leur importance par rapport au reste du corpus. Nous allons voir comment l'information structurelle est représentée ; comment les termes indexés sont-ils rattachés à cette structure ; et comment la structure influence-t-elle la pondération des termes ? Pour le processus d'indexation, le document est vu comme un arbre dont les nœuds représentent un élément et les feuilles les éléments de contenu.

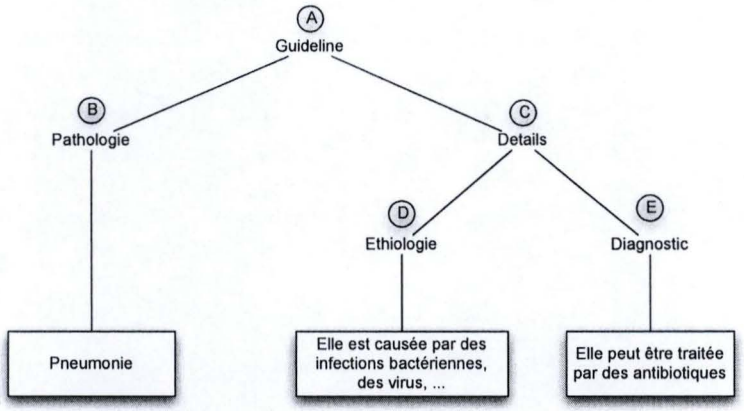


Figure 8 Document vu comme un arbre

Propagation de l'information

La première question qui se pose est de savoir si l'on doit ou non propager l'information contenue dans les feuilles aux nœuds parents pour l'indexation.

Une première approche appelée sous arbres imbriqués, suivie notamment dans (Schlieder & Meuss, 2002) et (Cui, Wen, & Chua, 2003), considère que chaque nœud représente un document atomique. Il faut alors constituer autant d'index qu'il n'y a de nœuds. Ainsi chaque index va représenter le contenu du sous-arbre, dont le nœud à représenter est la racine. En prenant le document représenté à la Figure 8, l'index représentant le nœud « A » représentera tous les termes du document initial ; l'index du nœud « B » contiendra le terme « pneumonie » ; l'index du nœud « C » comprendra les termes des nœuds « D » et « E », l'index du nœud « D » sera indexé sur base de son contenu, ... Cette solution a l'avantage de faciliter le processus de recherche, car elle traite les problèmes induits par la structure dès l'indexation ; la recherche dans un document structuré se réduit à une recherche dans plusieurs documents plats. Seulement comme les nœuds sont imbriqués les uns dans les autres, elle a l'inconvénient de stocker beaucoup d'informations redondantes.

Une autre approche, suivie notamment dans (Fuhr, 2001), (Gövert, Fuhr, Abolhassani, & Großjohann, 2003) et (Ogilvie & Callan, 2003), considèrent les nœuds comme des unités disjointes. « Le texte de chaque nœud de l'index est l'union d'une ou plus de ces parties disjointes » (Sauvagnat & Boughanem, 2009).

Mise en correspondance des termes avec la structure

L'emplacement des termes dans la structure du document peut être représenté de différentes manières. Nous allons exposer ici les trois principales approches exposées dans (Luk, Leong, Dillon, Chan, Croft, & Allan, 2002) : l'indexation basée sur les champs, l'indexation basée sur les chemins et l'indexation basée sur les arbres.

L'indexation basée sur les champs consiste à associer chaque terme au nom du nœud dans lequel il se trouve. On peut ainsi retrouver le nœud dans lequel le terme apparaît.

Terme	**Emplacement** (unités disjointes)
Pneumonie	Pathologie
Infection	Etiologie
Antibiotique	Diagnostic

L'indexation basée sur les chemins consiste à associer chaque terme au chemin permettant d'accéder au nœud contenant le terme. Le chemin d'accès peut être une requête XPath permettant d'accéder au nœud. Cette représentation permet d'accélérer les temps de traitement et lever l'ambiguïté lorsque plusieurs nœuds portent le même nom.

Terme	**Emplacement** (unités disjointes)
Pneumonie	Guideline/Pathologie
Infection	Guideline/Details/Etiologie
Antibiotique	Guideline/Details/Diagnostic

L'indexation basée sur les arbres nécessite de donner un identifiant unique à chaque nœud de l'arbre (comme dans la Figure 8). Les termes sont alors associés à cet identifiant.

Terme	**Emplacement** (unités disjointes)
Pneumonie	B
Infection	D
Antibiotique	E

Pondération des termes

Une fois que l'on a fourni un moyen de localiser les termes dans le document, il faut encore calculer une mesure pour représenter l'importance du terme. Les mesures de pondérations comme TF-IDF, utilisée en RI classique permette de montrer l'importance du terme dans le document et par rapport au reste de la collection. Ici, il faudrait aussi pouvoir rendre compte de l'importance du terme par rapport aux autres nœuds composant le document. Il faut aussi prendre en compte le fait que les termes importants sont moins souvent répétés dans les documents structurés. On trouve des adaptations du modèle vectoriel dans (Carmel, Efraty, Landau, Maarek, & Mass, 2002).

1.2.3. La mise en relation requête/document

En RI classique, le moyen privilégié pour permettre à l'utilisateur d'exprimer sa requête était d'utiliser un langage libre, sous la forme d'une liste de mots-clefs. La requête n'est formulée qu'en fonction du contenu, on parle de requête orientée contenu.

Les langages développés pour interroger les documents semi-structurés sont inspirés des langages développés pour les bases de données. Ceux-ci permettent surtout d'exprimer des conditions sur la structure des documents. Même si certains permettent aussi d'effectuer des recherches sur le contenu, ils sont plus rigoureux et exigent de préciser l'élément de la structure concerné. Ils demandent aussi à l'utilisateur d'exprimer la forme du résultat qu'il souhaite recevoir en réponse à la requête. Ces langages (XQuery, XPath, ...), ne sont donc pas adaptés à une approche RI où l'utilisateur n'a pas d'idée sur la forme de la structure des documents pouvant répondre à sa question. Les SRI proposent donc principalement des langages orientés contenus améliorés pour permettre à l'utilisateur d'exprimer des conditions sur la structure du résultat comme dans (Fuhr, 2001). D'autres vont jusqu'à proposer des requêtes mélangeant contenu et balise XML (Mass, Mandelbrod, Amitay, Carmel, Maarek, & Soffer, 2002) ; dans ce système, la requête :

```
<book>
        <author>John</author>
</book>
```

permet de retrouver le fragment de document :

```
<book>
        <fm><author><first>John</first></author></fm>
</book>
```

L'interprétation de la requête en RI, se fait différemment de ce qui se fait avec les langages natifs, orienté structure comme XQuery. De tels systèmes évaluent la requête de manière booléenne ce qui veut dire que soit une granule correspond parfaitement soit elle ne répond pas aux conditions exprimées dans la requête. Les SRI, eux utilisent des mesures de pertinences qui prennent aussi en compte les documents ne répondant que partiellement à la requête. Les modèles de RI traditionnelles ont donc été adaptés pour prendre en compte la structure des documents dans leurs mesures de la pertinence. On trouve des extensions du modèle vectoriel comme celle présentée dans (Meuss & Schlieder, 2002). Ce modèle permet d'effectuer des recherches dans des documents structurés avec des requêtes portant uniquement sur le contenu.

1.3. Représentation des connaissances

1.3.1. Introduction

La représentation de la connaissance en informatique a pour objectif de modéliser les connaissances que l'on a du monde dans un formalisme exploitable par un ordinateur. Cela se fait en ajoutant une dimension sémantique aux données. Il devient alors

possible de simuler le raisonnement de l'esprit humain par des programmes informatiques à l'aide de mécanismes d'inférences.

La représentation formelle de la connaissance s'est développée dans le domaine de l'intelligence artificielle (IA) qui lui consacre une branche à part entière : l'ingénierie des connaissances (IC). L'IC s'intéresse à l'acquisition, la modélisation et le stockage de connaissance en vue d'effectuer des raisonnements (semi-)automatiques sur ces connaissances. Les formalismes développés ne doivent pas seulement être compris par des programmes informatiques, mais rester compréhensibles pour des opérateurs humains.

Les ontologies sont couramment utilisées pour la représentation de connaissances. Elles définissent les éléments de base permettant de formaliser les connaissances utilisées en IC.

Les méthodes de Recherche d'Information exposée jusqu'ici représentent le contenu des granules sur base d'informations lexicales pour les approches traditionnelles (chapitre 1.1) et vont jusqu'au niveau syntaxique pour les approches issues de la RI structurée (chapitre 1.2). Les connaissances, une fois formalisées, permettent d'ajouter une dimension sémantique à la RI. Cela permet, notamment, de lever les ambiguïtés sur les termes utilisés pour l'indexation des documents et des interactions plus « intelligente » avec l'utilisateur.

Dans ce chapitre, nous verrons d'abord ce qu'est une ontologie. Nous verrons ensuite ce qu'est le Web sémantique, domaine qui a permis de propulser les recherches sur les ontologies. On s'intéressera ensuite aux différences entre les ontologies et les modèles conceptuels. On clôturera le chapitre sur les apports des ontologies en Recherche d'Information.

1.3.2. Ontologie

Initialement, l'ontologie est un concept philosophique. Le terme vient du grec « ontos », l'être et « logos », la science. Il désigne ainsi la science de l'être en tant qu'être (RICŒUR, 2009).

En informatique, et plus précisément en IA, les connaissances étaient d'abord représentées dans ce que l'on appelle des bases de connaissances. Une base de connaissances contient des faits et des règles. Ces règles, lorsqu'elles sont interprétées par un moteur d'inférence, permettent de simuler des raisonnements ; permettant de déduire de nouveaux faits à partir de faits existants. La limite de ce formalisme est qu'il n'est pas prévu pour être réutilisable.

Plus formellement, dans (Gruber, 1993), une ontologie est définie comme étant « une spécification explicite d'une conceptualisation ». Il définit le terme conceptualisation

comme étant « une vue abstraite, simplifiée du monde que l'on veut représenter pour une certaine raison ». Pour (Borst, 1997), les « ontologies sont une description formelle de la connaissance partagée d'un domaine ».

La définition la plus répandue rassemble les deux précédentes, ce qui donne : « Une ontologie est une spécification formelle, explicite d'une conceptualisation partagée » (Studer, Benjamins, & Fensel, 1998). Il précise aussi le rôle des termes clefs utilisé dans cette définition :

- « 'Conceptuel' se réfère à un modèle abstrait de certains phénomènes du monde en ayant identifié les concepts pertinents de ce phénomène. »
- « 'Explicite' signifie que les concepts utilisés et les contraintes sur leurs utilisations sont explicitement définis ». Les contraintes, comprennent aussi les relations entre concepts.
- « 'Formel' fait référence au fait que l'ontologie doit être compréhensible par une machine ». En d'autres termes, il faut utiliser un langage formel pour exprimer les concepts et les contraintes de l'ontologie ; ceci afin de permettre à une machine de les utiliser pour effectuer des raisonnements.
- « 'Partagée' reflète la notion de connaissance consensuelle décrite dans une ontologie, connaissance qui n'est donc pas propre à un individu, mais acceptée par un groupe ». Concrètement, cela signifie qu'une ontologie n'est pas conçue pour une application ou un objectif spécifique, mais doit être assez général que pour pouvoir être réemployée ; il s'agit d'une ressource partageable et réutilisable ; c'est un modèle de très haut niveau par opposition à un modèle conçu pour une application ou une technologie bien spécifique.

Les définitions que nous venons de voir nous apprennent qu'une ontologie est composée de *concept* et de *contraintes* sur ces derniers. La contrainte la plus souvent rencontrée étant la *relation* entre concepts.

Un *concept* représente une catégorie d'objet, une notion ou une idée (Gandon, 2006). Il est défini par généralisation des traits communs permettant d'identifier ce que l'on veut représenter. Les traits communs forment les propriétés du concept, que l'on appelle *intension*. Les individus pour lesquels ces propriétés s'appliquent sont appelés *extensions*. L'intension d'un concept généralise les propriétés communes à un ensemble d'individus alors que son extension fait référence aux individus du concept : ses instances.

Exemple : On veut représenter le concept de moto. Si l'intension de ce concept est « Véhicule à deux roues », on désigne aussi bien les vélos que les motos, car l'intension n'est pas assez précise que pour n'identifier que les motos. Si on y ajoute

l'intension « Véhicule possédant un moteur de plus de 125 cm^3 »[4], on ne désigne plus que les motos. L'intension « Véhicule à deux roues possédant un moteur de plus de 125 cm^3 » désigne bien le concept de moto. Une extension serait {Yamaha XJ 900, Suzuki VFR 800, Honda CBF 125, ...}.

Cet exemple montre bien que c'est l'intension qui caractérise le concept et non pas le terme utilisé pour le désigner, ici « moto » ; on pourrait aussi bien le désigner par le terme « bécane » ou l'image d'une moto. Cela permet d'éviter les problèmes d'ambiguïtés liés aux différents termes qui peuvent désigner un même concept ou a un même terme qui désigne différents concepts. On cherche donc à désigner un concept en décrivant sa signification, sa dénotation plutôt qu'en utilisant une référence lexicale.

Une ontologie permet d'organiser les concepts en hiérarchie. Il s'agit d'une forme un peu à part de relation, que l'on nomme « subsomption ». Ainsi dans l'exemple ci-dessous, le concept « Véhicule » subsume celui de véhicule « Motorisé » et véhicule « Non-Motorisé ». À l'inverse, on dira que « Motorisé » et « Non-Motorisé » sont subsumés par « Véhicule ».

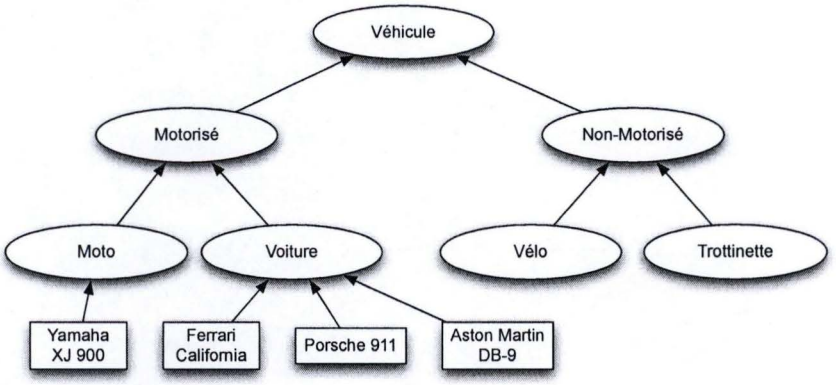


Figure 9 Exemple de hiérarchie de concepts

Tout comme en programmation orientée objet (POO), un concept B subsumé par un concept A signifie que B « est un » A ; B possèdent alors toutes les caractéristiques (relations et contraintes) de A par subsomption (héritage en POO) ; et A possèdent tous les individus (instances) de B. Ainsi en reprenant l'exemple d'ontologie représenté par la Figure 9, l'extension du concept véhicule motorisé est l'union de celle de Moto et Voiture c'est-à-dire Yamaha XJ 900, Ferrari California, Porsche 911, Aston Martin DB-9.

[4] Définition moto : http://www.linternaute.com/dictionnaire/fr/definition/moto/. Date :05/08/2011.

En plus des relations de subsomption, une ontologie permet de définir des relations sémantiques entre concepts ou entre individus. On peut par exemple définir une relation de composition entre véhicule et roue pour exprimer le fait qu'un véhicule possède des roues ou qu'un véhicule est conduit par un individu. Il est aussi possible de définir des liens de subsomption entre relations. Une relation, « est conduit par » entre les concepts Véhicule et Etre humain peut subsume la relation « est conduit par » entre les concepts Véhicule motorisé et Titulaire d'un permis de conduire ; ou encore entre l'individu titulaire du permis Paul et la voiture Porsche 911.

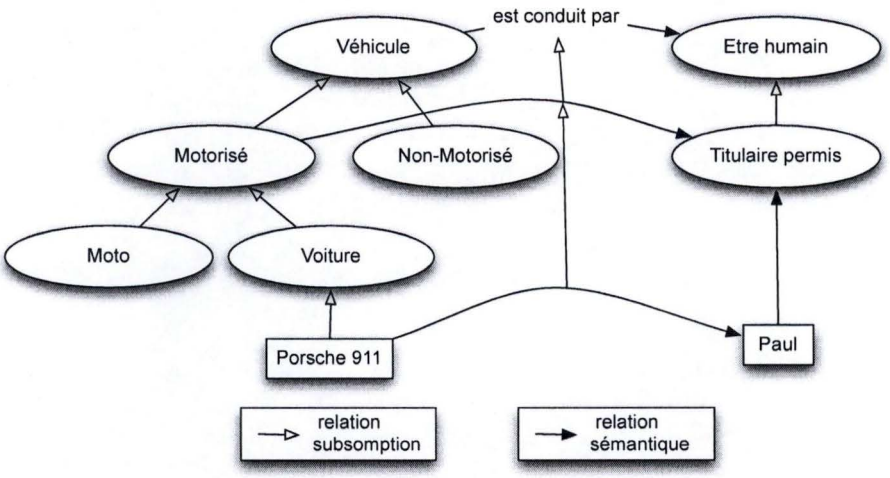


Figure 10 Exemple d'ontologie

Finalement, il est possible de définir différents types de contraintes (Gandon, 2006). On peut citer les contraintes d'intégrités permettant, par exemple, d'exprimer qu'un citoyen belge a un et un seul numéro d'identification au registre national. Spécifier la population d'un concept en fonction d'autres comme définir les individus d'un concept « SansAbris » comme étant les individus d'une classe « Habitant » qui n'ont aucune relation avec une classe « Logement ».

Les mécanismes permettant d'exprimer des contraintes sont plus ou moins riches en fonction du langage utilisé pour modéliser l'ontologie. Le chapitre suivant présente le langage le plus répandu, OWL, développé dans le cadre du Web sémantique.

1.3.3. Le Web Sémantique

Dans (Berners-Lee, Hendler, & Lassila, 2001), Tim Berners-Lee, connu pour être l'un des principaux inventeurs d'internet, expose sa vision du Web sémantique. Il s'agit d'ajouter une dimension sémantique au Web afin que des programmes informatiques puissent comprendre les informations qu'ils présentent à l'utilisateur. En effet, avec les technologies du Web traditionnel, le contenu d'un document n'est compréhensible que par un humain, les ordinateurs eux ne s'occupent que de la mise en pages ; bien que

des langages de balisage comme le HTML, leurs permettent de distinguer une entête, du corps du document ou encore de savoir quels sont les liens vers d'autres documents, ils ne donnent pas d'informations formelles sur la signification du contenu de ces éléments. De ce fait, les moteurs de recherches sur internet sont limités à la récolte de documents sur base des termes qu'ils contiennent ; ils utilisent les méthodes de Recherche d'Information non-structurée, vu au Chapitre 1.

L'idée derrière le Web sémantique n'est pas de remettre en cause les fondements du Web, mais d'enrichir l'existant, d'un système de métadonnées formelles, pour permettre à des machines d'effectuer des traitements automatiques sur les contenus actuels et à venir. À cette fin, le W3C[5] fondé par Tim Berners-Lee, a défini toute une famille de langages illustrés dans la figure ci-dessous.

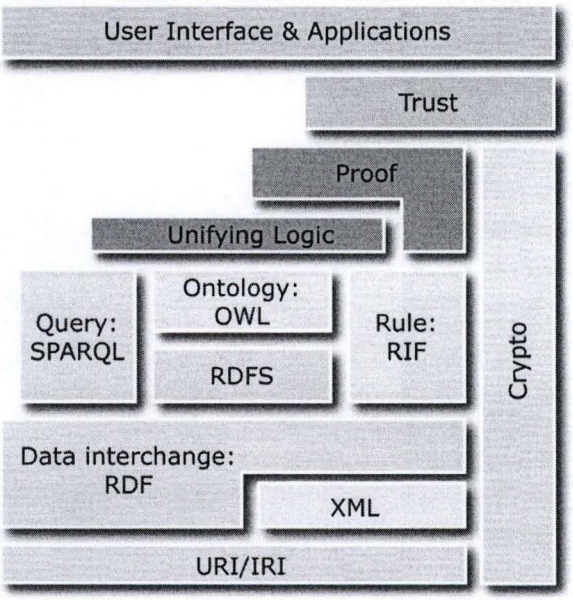


Figure 11 "Semantic Web Layer Cake"[6]

La première couche représente les URI et les IRI qui permettent d'identifier les ressources présentes sur le Web. Le langage de métadonnée, RDF (Ressource Description Framework), a été défini par le W3C pour décrire de façon formelle toutes les ressources identifiables par un URI. XML est utilisé pour sérialiser RDF. RDFS et OWL sont des langages permettant de créer des ontologies. SPARQL est un langage permettant d'interroger RDF. Aucun standard n'a encore été défini pour les autres couches. À titre d'information, la couche logique doit permettre de définir des règles

[5] W3C : http://www.w3c.com. Date : 06/08/2011.

[6] Représentation des Connaissances et Raisonnement : http://www.fadi.lautre.net/cours/mines3aWS/03-RCR.xhtml. Date : 06/08/2011.

pour réaliser des inférences. La couche « preuve » doit permettre à l'utilisateur de comprendre comment et d'où ont été déduits les faits présentés par la couche logique. La couche « confiance » comprendra notamment des mécanismes de signatures digitales. Finalement, la couche « Utilisateurs et Applications » représentera toutes les applications qui utilisent les technologies du Web Sémantique c'est-à-dire des sites Web, des agents fournissant des services, ...

RDF

RDF (Ressource Description Framework) est un langage formel de description de ressource présente sur le Web (W3C, 2004). Le fait qu'il s'agisse d'un langage formel permet d'appliquer des traitements automatiques sur les descriptions ; description qui présente les ressources sous forme de graphe.

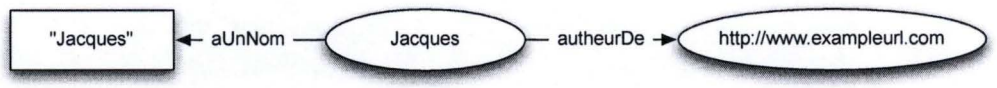


Figure 12 Graphe RDF

Un graphe RDF est décrit par un ensemble de déclarations. Une déclaration est un triplet combinant :

- Un sujet, référençant l'URI de la ressource à décrire (ex. : http://www.example.com/Staff/Jacques) ;
- Un prédicat, donnant le type de la relation, appelé propriété en RDF, applicable à cette ressource (ex. : autheurDe) ;
- Un objet, qui est la valeur de la propriété (ex. : http://www.exempleurl.com).

L'objet peut désigner une ressource ou une valeur, appelée littéral. Un littéral prend la forme d'une chaine de caractère.

Le graphe de la Figure 12 sera décrit par les triplets :

- (http://www.exampleurl.com/Staff/Jacques, aUnNom, « Jacques ») et
- (http://www.exampleurl.com/Staff/Jacques, autheurDe, http://www.exempleurl.com).

Une ressource désigne toute « chose qui peut être identifiée sur le Web, même quand il n'est pas possible de la récupérer directement. » (W3C, 2004). Cela signifie qu'une ressource désigne aussi bien une entité concrète comme une page web, une donnée multimédia, ... qu'une entité abstraite comme une personne ou tout autre objet du

monde réel. L'important est que ces ressources puissent être identifiées par une URI[7] (Uniform Ressource Identifier).

Différents formats comme RDF/XML, NTriple, Notation3 ou encore Turtle existent pour la sérialisation de graphe RDF. RDF/XML est le langage par défaut, défini par le W3C. Il est basé sur une syntaxe XML comme on peut le voir dans l'exemple ci-dessous, décrivant le graphe de la Figure 12 en RDF/XML.

```
<rdf:Description rdf:about="http://www.exampleurl.com/Staff/Jacques">
        <aUnNom>Jacques</aunNom >
        <autheurDe rdf :ressource="http://www.exampleurl.com" />
</rdf:Description>
```

Figure 13 Extrait d'un document RDF/XML

RDF Schéma

RDF Schéma (RDFS) (W3C, 2004) est un langage de description de vocabulaire RDF. Il ajoute des constructions à RDF permettant :

- la définition de classes
- la définition de propriétés
- les relations de subsomptions entre classes ou entre propriétés.

Il est ainsi possible de représenter des ontologies. Pour cela, RDFS définit différents types de ressources RDF : les classes, les propriétés et les littéraux. Il définit aussi différents types de propriétés : les domaines, les intervalles, les types, les relations de sous-classe, la relation de sous-propriété et les labels.

[7] Un URI est une chaine de caractère permettant d'identifier toutes ressources sur le Web qu'elles soient physiques ou abstraites.

```
@prefix :    <http://www.exampleurl.com/sample.rdfs#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.

:Moto                rdfs:subClassOf    :Motorise.
:Voiture             rdfs:subClassOf    :Motorise.
:TitulairePermis     rdfs:subClassOf    :EtreHumain

:estConduitPar       rdfs:range     :Motorise ;
                     rdfs:domain    :TitulairePermis.

:Paul       a   :TitulairePermis ;
:Porche_911 a   :Voiture ;
            :estConduitPar  :Paul.
```

Figure 14 RDF Schéma utilisant le format Turtle, plus lisible que RDF/XML

Ces nouvelles primitives permettent de déclarer des concepts (classe en RDFS) comme « Moto », « Voiture » ou « TitulairePermis » ; de déclarer qu'une classe est subsumée par une autre en utilisant le prédicat « rdf:subClassOf » ; de déclarer une ressource comme instance d'une classe comme le fait que « Paul » est un « Titulaire du Permis ». On peut aussi définir des propriétés comme « estConduitPar» ; définir le domaine de valeur de cette propriété comme étant la classe des véhicules « Motorisé » ; et l'intervalle de cette propriété comme étant les « Titulaire du Permis » ; définir des relations de subsomption entre propriétés grâce au prédicat 'rdf:subPropertyOf'. Finalement, un label est un prédicat permettant de lier une ressource à un littéral pour en donner une description en langage naturel.

OWL

OWL, pour langage d'ontologie sur le web, est le langage défini par le W3C pour représenter de la connaissance sur le Web. Il a été conçu pour combler le manque d'expressivité de RDFS. Il permet ainsi de définir de nouveaux vocabulaires.

Ainsi, OWL permet de définir des contraintes plus poussées ; comme le fait que la population d'une classe soit composée de l'intersection, l'union ou le complément d'autres classes ; qu'une propriété est transitive, fonctionnelle ou l'inverse d'une autre propriété ; ou encore définir une classe en énumérant ses objets ; on peut aussi restreindre la valeur d'une propriété pour une classe ce qui permet par exemple de lié les individus d'une classe « Habitant » à une autre classe « SansAbris » quand la propriété « adresse » n'a pas de valeur.

Le problème est que certaines constructions peuvent rendre les inférences OWL indécidables (Horrocks, Patel-Schneider, & Harmelen, 2003). Les assertions sur des chaines de propriété permettant d'exprimer qu'un oncle est le frère d'un parent sont un exemple de constructions qui rendent les inférences d'OWL indécidable. Afin de pouvoir garder son pouvoir d'expressivité, garantir un temps d'exécution optimal et rester compatible avec RDFS, il a été décidé de le scinder OWL en trois niveaux.

OWL Full est le langage complet. Il est pleinement compatible avec RDF et RDFS. Son pouvoir d'expressivité élevé le rend indécidable.

OWL DL, pour OWL Description Logic, est une version d'OWL qui utilise un langage abstrait basé sur les frames et la logique descriptive. Il n'est donc pas compatible avec RDF(S). OWL DL ne permet pas certaines constructions, comme des relations entre chaines de propriétés, pour rester décidable. La complexité dans le pire des cas est exponentielle sur une machine non déterministe.

OWL Lite, un sous ensemble d'OWL DL, utilise un sous-ensemble de la syntaxe pour une plus grande simplicité et garantir que les inférences soit dans le pire des cas, de complexité exponentielle déterministe. Il n'est par contre plus possible de définir une classe comme l'union ou le complément d'autres classes, les contraintes de cardinalités sont limités à zéro ou une, ...

SPARQL

SPARQL, « Simple Protocol and RDF Query Language », est un langage de requête pour RDF recommandé par le W3C (W3C, 2008). C'est une sorte de SQL, duquel il est d'ailleurs inspiré, pour l'interrogation de base de données RDF.

Une requête SPARQL comprend une série de clauses inspirée de SQL : une partie « FROM » pour spécifier quelle base de données RDF doit être interrogée ; une partie « SELECT » permettant de définir la forme du résultat ; une partie « WHERE » permettant de formuler des conditions en utilisant le modèle de triplet RDF ; une partie « ORDER BY » pour spécifier d'éventuelles contraintes sur le classement des résultats. Une requête SPARQL comprend aussi une entête réservée aux déclarations de préfixes.

La requête suivante nous permettrait d'avoir la liste des conducteurs et de leurs voitures pour l'ontologie définie à la Figure 14.

```
PREFIX abc:<http://www.example.com/exampleOntology#>
SELECT ?conducteur ?voiture
WHERE {
        ?voiture abc :estConduitPar ?conducteur .
}
```

Figure 15 Exemple de requête SPARQL

1.3.4. Ontologies et modèles conceptuels

Les ontologies présentent des similitudes avec les schémas conceptuels : ce sont deux langages permettant de modéliser des concepts du monde réel à l'aide de classe, de propriétés et de relations. Ces similitudes sont exploitées, notamment, dans des travaux visant à créer des bases de données à partir d'ontologie du domaine ou encore pour donner du sens à une base de données.

Cependant, ces deux formalismes divergent sur de nombreux points, que l'on retrouve dans (Fankam, Bellatreche, Hondjack, Ameur, & Pierra, 2009) qui a inspirés la rédaction de cette section ; ils partent de la définition proposée dans (Jean, Pierra, & Ait-Ameur, 2007) pour marquer ces différences :

> *« Une ontologie est une représentation formelle, référençable et consensuelle de l'ensemble des concepts partagés d'un domaine sous forme de classes, de propriétés et de relations qui les lient. »*

Ainsi, les ontologies utilisent des langages *formels* pour permettre à des machines de réaliser des inférences pour tirer de nouveaux faits à partir des connaissances qui y sont représentées. Les modèles conceptuels modélisent les données utiles à une application et uniquement celle-là ; ces modèles suivent l'hypothèse du monde clos en imposant aux données de respecter toutes les contraintes qui y sont définies ; les ontologies font l'hypothèse inverse (celle du monde ouvert), une instance peut violer une contrainte définie dans l'ontologie, car on suppose que les informations qu'elle contient peuvent être incomplètes.

De plus, les ontologies présentent une vision *consensuelle* d'une réalité ; c'est à dire admises par un groupe de personnes. Alors qu'un modèle conceptuel est réalisé selon la vision de son concepteur pour une application précise. Un modèle conceptuel contrairement à une ontologie est prévu pour un contexte bien spécifique et n'est pas réutilisable. Une ontologie, au contraire, doit décrire les concepts du domaine indépendamment de tout contexte applicatif ; et ne dois pas seulement refléter la vision de son concepteur, mais plutôt d'un groupe d'experts du domaine modélisé. En résumé, l'ontologie se place à un niveau d'abstraction plus élevé que celui utilisé dans les modèles conceptuels. Cela permets d'utiliser une ontologie comme artefact

préliminaire à la construction d'un schéma conceptuel comme dans (Conesa & Olivé, 2006) et (Spaccapietra, Parent, Cullot, & Vangenot, 2004) ; ou encore pour ajouter une dimension sémantique à des schémas conceptuels hétérogènes (An, Borgida, & Mylopoulos, 2006).

Finalement, les modèles conceptuels s'arrêtent à la représentation des concepts alors que les ontologies permettent d'aller plus loin en instanciant ces concepts dans un même modèle (Hainaut, 2009).

1.3.5. Apports des ontologies en Recherche d'Information

En Recherche d'Information classique, le contenu d'un document est représenté par un index constitué d'une liste de termes. Un terme est une chaine de caractère faisant référence à une entité du monde réel. Cette liste de termes ne contient pas d'informations sur les relations sémantiques qui les lient. Ainsi, un mot peut désigner différents concepts (polysémie) et un concept peut être décrit par différents mots (synonymie), sans que cela apparaisse dans l'index. Si un utilisateur utilise un certain terme pour effectuer sa recherche, il ne trouvera pas les documents qui contiennent les synonymes de ces termes.

Afin d'améliorer les performances de ces SRI, de nouvelles méthodes sont apparues, où le contenu d'un document n'est plus représenté par une liste de termes qu'il contient, mais sur base des concepts que ces termes représentent. Ces concepts sont modélisés dans une ontologie qui contient aussi les relations qu'ils partagent ; comme nous le verrons, la formalisation des relations sémantiques dans l'ontologie apporte beaucoup au processus de recherche.

L'indexation sémantique se base sur l'intuition selon laquelle « le sens d'un texte (des mots) dépend des relations conceptuelles entre les objets du monde auxquels ils font référence plutôt que des relations linguistiques et contextuelles trouvées dans les textes » (Haav & Lubi, 2001).

Cette intuition se traduit par des mesures de similarités sémantiques entre concepts d'une ontologie. On trouve différentes mesures dans (Lin, 1998), basée sur les suppositions suivantes : Plus deux concepts partagent des caractéristiques, plus ils seront similaires ; à l'inverse plus leurs caractéristiques divergent, moins ils seront similaires ; finalement, l'indice de similarité maximum ne peut être atteint que lorsque les concepts sont identiques.

Afin d'indexer les documents, une ontologie modélisant le domaine d'activité du corpus est nécessaire. Une première solution consiste à construire l'ontologie à partir du ou des corpus indexés (Toussaint, 2004). L'avantage est que le corpus et l'ontologie sont en parfaite adéquation. Le principal problème de cette approche est de perdre du temps pour réaliser une ontologie plutôt que d'en réutiliser une existante.

Une autre solution est de réutiliser des ressources existantes ; on peut utiliser un thésaurus que l'on transforme en une ontologie (Assem, Menken, Schreiber, Wielemaker, & Wielinga, 2004) ; ou réutiliser une ontologie préexistante (Baziz, Boughanem, Aussenac-Gilles, & Chrisment, 2005) (Vallet, Fernández, & Castells, 2005). La deuxième solution parait plus appropriée, car elle ne dénature pas la philosophie derrière les ontologies (chapitre 1.3.2) qui est de représenter de la connaissance indépendamment de toutes applications. En effet, les ontologies créées à partir d'un corpus ne sont généralement utilisables qu'avec ce corpus.

Dans la suite, nous allons voir comment les relations hiérarchiques et sémantiques, entre les concepts d'une ontologie, sont exploitées pour améliorer les différentes étapes du processus de RI : l'indexation, l'appariement requête/document et la reformulation de la requête.

Indexation sémantique

L'indexation sémantique d'un document comprend deux grandes étapes :

- L'identification des concepts, qui vise à mettre en correspondance les concepts de l'ontologie avec le document.
- La pondération, permettant de refléter l'importance de ces concepts dans le document.

L'identification des concepts dans le document peut se faire manuellement ou automatiquement. Lorsqu'elle est automatique, on se sert des labels, décrivant les concepts de l'ontologie en langage naturel, pour identifier les concepts dans le document. On commence par extraire les termes du document en suivant les mêmes étapes que pour la construction d'un index terminologique (analyse lexicale, élimination des mots fonctionnels, lemmatisation [voir chapitre 1.1.2]).

Une fois les termes extraits, on cherche à les mettre en correspondance avec les labels décrivant les concepts de l'ontologie. Pour cela, il faut pouvoir reconnaitre les groupes de termes qui forment un syntagme comme « Vers de terre » ; la stratégie décrite dans (Baziz, Boughanem, Aussenac-Gilles, & Chrisment, 2005) consiste à privilégier les labels les plus longs. Il faut aussi résoudre les problèmes d'ambiguïté survenant lorsqu'un label correspond à plusieurs concepts ; dans ce cas, plusieurs stratégies peuvent être utilisées et éventuellement combinées. Une première stratégie consiste à choisir des combinaisons de concepts sémantiquement proches comme expliqué dans (Navigli & Velardi, 2003). Dans (Guha, McCool, & Miller, 2003), ils exposent trois critères permettant de choisir le concept le plus probable : la fréquence d'apparition du concept dans le corpus, le profil de l'utilisateur et l'historique de recherche.

Lorsque les concepts ont été identifiés, on calcule un indice de pondération pour refléter leurs importances dans le document. Une première approche consiste à utiliser

des mesures statistiques similaires à celles utilisées pour l'indexation terminologique (Vallet, Fernández, & Castells, 2005). Une autre approche consiste à mesurer un indice de similarité entre les concepts indexés, pour la pondération.

Mise en relation requête/document

En plus des langages utilisés par les SRI traditionnels (chapitre 1.1.3), il est aussi possible d'utiliser des langages propres aux ontologies. Cela permet de profiter des mécanismes d'inférences que proposent les moteurs interprétant ces langages. En général, l'utilisateur exprime sa requête en langage naturel et le SRI la traduit dans un langage d'interrogation propre aux ontologies ; comme dans (Guha, McCool, & Miller, 2003) où ils transforment les requêtes utilisateurs dans le langage GetData qu'ils ont eux-mêmes défini pour interroger l'ontologie. D'autres approches proposent des systèmes graphiques où l'utilisateur sélectionne des concepts en naviguant dans l'ontologie. La figure ci-dessous, tirée de (Fluit, Sabou, & Harmelen, 2003), donne un exemple d'interface, intitulée « Cluster Map ». Elle permet aussi de visualiser graphiquement la pertinence des documents par rapport à la requête. La partie de gauche affiche une liste de concepts parmi lesquels l'utilisateur fait son choix ; la partie de droite représente les documents repartis en cluster en fonction des concepts qu'ils partagent avec la requête.

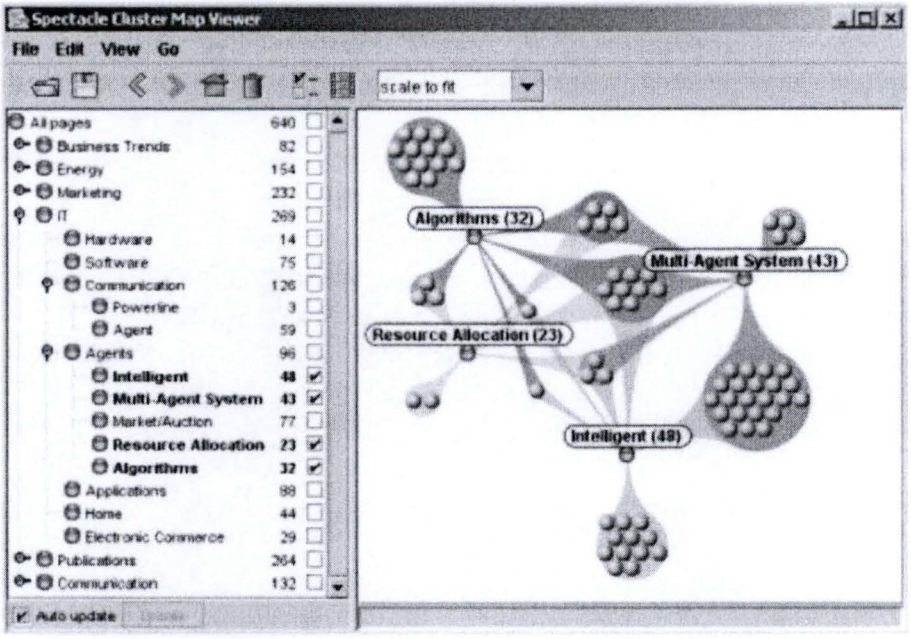


Figure 16 Prototype d'interface permettant de sélectionner les concepts de l'ontologie

Lorsque les documents et la requête sont indexés à partir de la même ontologie, les relations sémantiques entre les concepts peuvent être utilisées pour calculer l'indice de similarité entre la requête et les documents ; cela permets de trier la pertinence des

résultats comme en RI classique. Par exemple dans (Andreasen, Bulskov, & Knappe, 2003), une mesure part du principe que si deux concepts sont subsumés par un même concept, ils sont plus similaires, leur indice de similarité sera d'autant plus élevé qu'ils sont proches de ce concept ; une autre mesure se base sur le nombre de nœuds dans l'ontologie qui peuvent être atteints par les deux nœuds via n'importe quel type de relations sémantiques.

Reformulation de la requête

Les relations hiérarchiques et sémantiques entre concepts dans une ontologie peuvent être exploitées pour enrichir la requête de nouveau concept proche de ceux initialement visés par l'utilisateur.

L'ontologie peut remplacer un thésaurus en RI classique (voir chapitre 1.1.4) ; dans ce cas, il faut d'abord désambiguïser les termes de la requête initiale pour trouver les concepts correspondants dans l'ontologie ; il faut donc obligatoirement que des labels donnent une description en langage naturel des concepts. Une fois les concepts identifiés, on peut comme dans (Lawarrée, 2010), trouver de nouveaux concepts selon trois types d'enrichissement : par généralisation, par spécialisation et via les relations sémantiques. L'enrichissement par généralisation permet de trouver des concepts plus généraux, les ancêtres du concept à enrichir. L'enrichissement par spécialisation permet de trouver des concepts plus spécifiques, les descendants du concept à enrichir. L'enrichissement par relation sémantique permet de trouver des concepts liés au concept à enrichir selon une relation choisie par l'utilisateur.

Chapitre 2

Modélisation de données hétérogènes

2.1. Introduction

Ce chapitre aborde la représentation de descriptions de documents et données hétérogènes dans une base de données commune, surnommée métabase. Une description est composée d'un index représentant le contenu du document et d'un modèle décrivant sa structure.

La base de données qui est utilisée pour stocker ces descriptions a été développée dans le cadre du projet GISELE. Elle a été employée lors d'un stage au Cetic, pour expérimenter les possibilités de la réutiliser pour stocker toutes sortes de descriptions relatives aux activités et ressources échangées par la plateforme e-Health. Elle a notamment servi à enregistrer des descriptions d'itinéraire de soins cliniques sous la forme de workflow.

Nous allons voir comment on peut y consigner les informations relatives à différents types de documents, que l'on classera selon le niveau de granularité qu'ils permettent d'atteindre ; en partant du moins structuré, on a :

- les documents non structurés ou « plats » (documents textes) ;
- les documents semi-structurés (documents XML) ;
- les données structurées (base de données, ontologies, ...).

La structure d'un document est généralement décrite par un modèle. Il s'agit d'un modèle conceptuel et logique dans le cas d'une base de données, une DTD ou un XML Schema pour un document XML, ...

Utiliser une base de données et donc un modèle commun va nous permettre de représenter les correspondances qui peuvent exister entre ces descriptions. Il y a correspondance entre composants de descriptions lorsque ces composants représentent les mêmes entités du monde réel. On peut lier les concepts d'une ontologie aux concepts correspondants dans un schéma conceptuel afin d'ajouter une dimension sémantique à ce dernier. Il y a aussi correspondance lorsqu'un modèle est issu de la transformation d'un autre, par exemple entre un schéma logique et un schéma conceptuel. Dans le premier cas, on parlera de correspondances horizontales et de correspondances verticales dans le deuxième cas.

Pour la suite de ce chapitre, nous allons découvrir la métabase que nous avons réutilisée pour encoder nos descriptions. Dans la deuxième section, nous parlerons des différentes manières de modéliser ces descriptions dans la métabase. Les approches qui y sont présentées ne se limitent pas au cas de la métabase GISELE et peuvent servir de point de départ pour le développement d'un tout nouveau méta-repository. On s'intéressera ensuite à la représentation, dans la métabase, de documents de plus en plus structurés. Nous commencerons par les documents non structurés avec les textes plats, les documents semi-structurés avec les documents XML et les données structurées avec les bases de données. Finalement, nous nous intéresserons à la modélisation des correspondances verticales puis horizontales entre composants inter-schéma.

Dans ce chapitre, on suppose que les descriptions des documents existent et on ne commentera pas la façon dont ceux-ci sont indexés.

2.2. La base de métadonnées

La métabase utilisée pour représenter les descriptions de source de donnée, décrite dans (Hainaut, Brogneaux, & Cleve, Base de modèles pour les itinéraires de soins, 2010), a été développée dans le cadre du projet de GISELE. Ce projet est « consacré à la modélisation et à la validation d'itinéraires de soins en cancérologie »[8]. Un itinéraire de soins étant formalisé par un « workflow » (processus). La base de données GISELE a donc été développée pour ce type d'artefact autour duquel gravitent différents éléments nécessaires à leurs exploitations. Ainsi, un processus évolue dans une organisation et utilise des informations ; il nécessite ou produit des ressources ; l'accès aux processus, informations et ressources doit être contrôlé. Ces différents aspects, liés aux processus, ont été répartis dans cinq sous-systèmes composant le schéma conceptuel de la métabase.

[8] (Hainaut, Brogneaux, & Cleve, Base de modèles pour les itinéraires de soins, 2010)

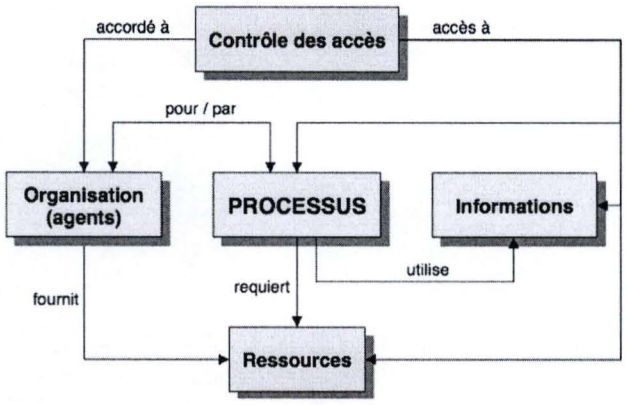


Figure 17 Les différents systèmes du modèle GISELE

Deux parties du modèle conceptuel de la métabase GISELE nous intéressent particulièrement : son système « méta » et son système consacré à la modélisation d'informations.

Le sous-système « méta » est constitué d'objets génériques qui reprennent les caractéristiques communes des objets plus spécifiques, répartis dans les cinq sous-systèmes du modèle. Ces objets génériques permettent aussi de définir de nouveaux objets qui seraient absents de la définition initiale du domaine d'activité, sans avoir à modifier le schéma de la base de données.

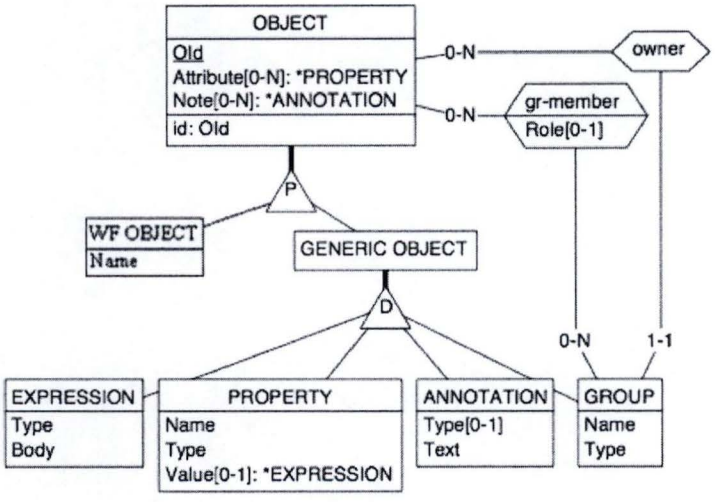


Figure 18 Schéma ERA du sous-système « META »

Un **objet** possède un identifiant absolu (oid) obligatoire, un ensemble d'attributs et un ensemble de notes. Toutes les entités du schéma descendent de l'entité « **object** » ; via l'entité « **WF Object** » pour les objets spécifiques ; via l'entité « **Generic Object** » pour les objets génériques : « **Expression** », « **Property** », « **Annontation** » et

« **Group** ». Une **expression** est caractérisée par un type (Type) et un corps (Body) contenant une expression dans un langage formel (formule mathématique, logique du premier ordre, ...). Une **propriété** (Property) est caractérisée par un nom, un type et une éventuelle valeur de type « Expression ». Une **annotation** est caractérisée par un éventuel type et un texte en langage naturel. Un **groupe** est caractérisé par un nom et un type. Il représente un agrégat d'objets, ses membres. Un membre (gr-member) est une association entre un groupe et un objet dont on peut préciser le rôle dans cet agrégat d'objet. Un groupe appartient à un objet (owner). Un objet de Workflow (WF object) abstrait les caractéristiques communes aux entités spécifiques, utiles au domaine des workflow, dont font partie les entités du sous-système destiné à la modélisation des informations. Les entités spécifiques, dont l'entité d'information détaillée ci-dessous, héritent donc des caractéristiques de cette entité.

La partie du schéma conceptuel dédiée à la représentation des informations contient des entités qui vont nous permettre d'enregistrer des descriptions de source de données. Cette partie a été conçue pour représenter les informations utilisées par un processus. Ces informations pouvant prendre des formes très diverses, les entités destinées à la représenter sont très génériques.

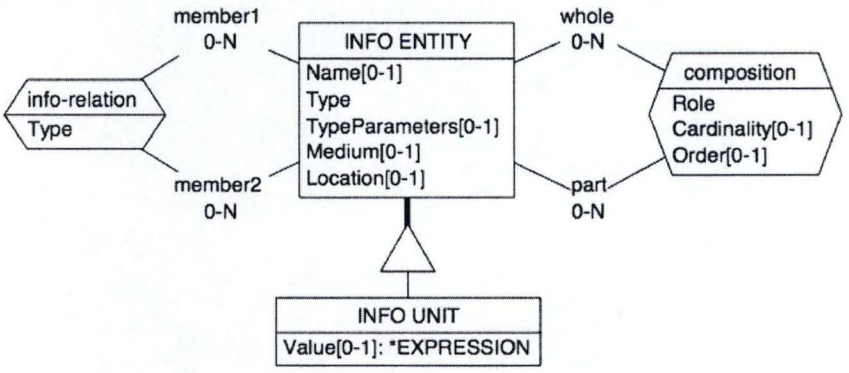


Figure 19 – Sous-ensemble du schéma conceptuel GISELE pour la modélisation d'information[9]

L'**entité d'information** est le concept central de ce sous-schéma. Il est destiné à représenter n'importe quel type de donnée ou d'informations. Une entité d'information peut posséder un nom (Name). Elle possède un type (Type) auquel on peut associer des paramètres (TypeParameters). Elle peut être enregistrée sur un support (Medium) quelconque et peut être accessible à une certaine adresse (Location). On peut, par exemple, s'en servir pour représenter un document numérique ou une base de donnée, accessible à l'URL renseignée par l'attribut « Location ».

[9] (Hainaut, Brogneaux, & Cleve, Base de modèles pour les itinéraires de soins, 2010)

Une **unité d'information** est une entité atomique, c'est-à-dire sans relation de composition avec une autre entité. C'est une entité d'information qui peut avoir une valeur sous la forme d'une **expression**.

En composant les entités d'information, on peut représenter des données plus complexes. Le type d'association « **composé** » permet de représenter les relations entre un agrégat (whole) et ses composants (part). Chaque relation de composition définit le rôle (Role) du composant. Il est possible de préciser le nombre d'instances de ces composants qui sont permises pour une instance de l'agrégat (Cardinality). Finalement, un numéro d'ordre (Order) peut être attribué pour déterminer la place du composant dans le composé. Ce type d'associations servira, par exemple, à représenter un schéma de base de données composée de tables. Chaque table étant composée d'attributs et d'identifiants. On peut aussi l'utiliser pour modéliser un document XML composé d'éléments et d'attributs, ...

D'autres types de relations peuvent exister entre entités d'information. Une « **info-relation** » représente une relation d'un certain type (Type) entre entités d'information.

2.3. Représentation des descriptions

Nous allons discuter ici de différentes approches permettant d'enregistrer des descriptions de sources de données, définis dans différents modèles, dans une base de données. Nous distinguons trois techniques différentes pour modéliser une description :

- comme une suite de données binaire
- selon sa syntaxe
- selon sa sémantique

Description vue comme une suite de données binaire

L'encodage brut est la façon la plus simple d'encoder une description. Cette méthode consiste à copier le document, contenant la description, bit à bit dans une colonne de la base de donnée sous forme de BLOB. Cette méthode ne permet pas de n'exploiter qu'une partie du contenu de la description ; pour cela, il faut en récupérer l'entièreté. Cela implique qu'il n'est pas non plus possible de représenter les correspondances entre les granules issues de modèles différents. On préférera cette approche pour sa facilité d'implémentation lorsque l'on souhaite enregistrer une description qui ne doit pas être mise en correspondance avec d'autres.

Modélisation selon sa syntaxe

La deuxième solution nécessite d'ajouter au schéma de la métabase, des structures permettant de représenter la syntaxe du langage utilisé pour la sérialisation de la description à encoder. Il est alors possible d'enregistrer des instances de ce modèle encodées selon cette syntaxe. Seulement, pour certains modèles comme RDF, il existe plusieurs syntaxes. Une ontologie sérialisée en utilisant la syntaxe RDF/XML et une autre utilisant la syntaxe N3 sera alors représentée différemment dans la métabase. Les processus d'enregistrement et d'exploitations seront différents selon que l'ontologie RDF soit décrite avec la syntaxe RDF/XML ou avec la syntaxe N3. Le cas d'UML est plus problématique, car il n'existe pas de syntaxe couramment utilisée bien que la norme XMI[10] ait été introduite par l'OMG pour l'échange de schémas UML. Cette approche peut être utilisée si :

- il est compliqué de s'abstraire de la syntaxe utilisée par ce modèle ; c'est souvent le cas lorsque le modèle est dépendant des technologies utilisées pour l'implémenter comme un schéma XML ou logique.
- on souhaite pouvoir charger qu'une partie de la description.
- on veut pouvoir réaliser des liaisons avec les composants d'autres descriptions ; tout en profitant de l'intégrité référentielle qui garantit que les mappings restent cohérents si une des descriptions est modifiée.

Modélisation selon la sémantique

La solution aux inconvénients de la précédente approche est l'encodage selon la sémantique du modèle. Elle consiste à modéliser les concepts du langage de modélisation indépendamment des syntaxes existantes pour sa sérialisation. On se basera généralement sur le méta-modèle du langage pour modéliser ses instances dans la métabase.

Pour la mise en œuvre de cette solution, on peut modéliser les concepts d'un langage de modélisation existant ou redéfinir un nouveau modèle destiné à encoder la sémantique d'une classe de langage de modélisation existant. Par classe, on entend un ensemble de langage de modélisation définis sur des concepts équivalents et utilisés pour une même finalité. On peut citer la classe des modèles conceptuels comprenant le modèle ERA, Merise, UML, ... ; la classe des modèles de workflow comprenant UML, BPMN, Little-JIL, ... ; ou encore les langages d'ontologie comprenant RDFS, OWL, SKOS, ...

Si l'on prend pour exemple, le cas de la représentation des schémas conceptuels dans la métabase. En utilisant la première approche, c'est à dire en utilisant un modèle

[10] XMI : http://www.omg.org/spec/XMI/. Date : 23/07/2011.

conceptuel existant comme ERA, on modélisera uniquement les concepts présents dans le métamodèle ERA. La modélisation de la sémantique d'une classe de modèle, consiste à créer un modèle générique pour représenter les modèles conceptuels, indépendamment du formalise utilisé dans les modèles existants ; on utilisera alors des noms neutres pour désigner les éléments issus de ses modèles. Il faudra aussi prendre en compte l'ensemble des constructions permises par ces différents modèles ; le but étant de pouvoir représenter plusieurs langages de modélisation à l'aide d'un même modèle. Pour réaliser cette approche, il faut donc que les concepts appartenant à ses langages soient sémantiques proches. Créer un modèle commun pour les schémas conceptuels, implémenté par le modèle ERA, le MCD de Merise et le modèle objet d'UML est réalisable parce que les concepts faisant partie de ses modèles représentent la même chose ; le concept d'objet dans les diagrammes de classe à la même signification qu'une entité ERA ou Mérise si le diagramme modélise un schéma conceptuel de donnée.

Dans la suite, nous allons voir comment nous avons enregistré les descriptions de données hétérogènes. Nous avons à chaque fois essayé de nous abstraire de toutes syntaxes pour nous concentrer sur la sémantique. Nous n'avons pas modifié le schéma de la métabase GISELE pour modéliser ces différentes descriptions ; préférant les représenter en utilisant les structures existantes, présentées au chapitre précédent.

2.3.1. Formalisme utilisé pour les règles de transformations

Dans la suite, nous allons voir comment représenter les descriptions dans la métabase. Pour cela, nous allons définir des règles permettant de transformer une description exprimée dans un modèle source en une description utilisant les constructions du modèle de la métabase (modèle cible), vue au chapitre 2.2.

Afin de définir ces règles, nous avons utilisé un pseudo-langage dans un style déclaratif. Ce langage utilise le vocabulaire défini par le modèle source et cible de la transformation. En plus, on définit les opérateurs suivants :

- Un opérateur d'affectation par référence (« = »), utilisé dans des expressions du type : *mavariable* = $TE(attr_1, \ldots, attr_n)$; il permet d'utiliser *mavariable* pour faire référence à l'entité *TE*.
- Un opérateur permettant de naviguer dans le schéma conceptuel (« . ») ; si *ent1* référence une entité possédant un attribut *attr1* et un rôle *role1* alors l'expression *ent1.attr1* désigne cet attribut et *ent1.role1* se rôle.
- Un opérateur de concaténation (« · ») qui permet de concaténer des chaines de caractère ; ainsi 'Base' · ' de ' · 'données' est équivalent à 'Base de données'.

Ce pseudo-langage doit permettre de décrire les constructions d'un schéma Entité-Relation-Attribut (ERA). Dans la figure ci-dessous, on voit que chaque élément graphique d'un modèle ERA peut être représenté en pseudo-code.

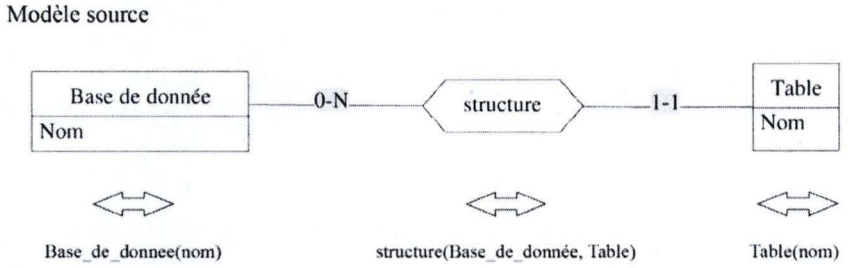


Figure 20 Exemple de modèle d'origine

Ainsi, une table *TABLE* est vue comme une relation qui agrègent ses attributs t_1, ..., t_n et peut être représentée par la notation $TABLE(t_1, ..., t_n)$. Une association *ASSOC* est une relation qui agrègent ses éventuels attributs t_1, ..., t_n et ses rôles r_1, ..., r_n et peut être représentée par la notation $ASSOC(t_1, ..., t_n, r_1, ..., r_n)$.

Le but de ce pseudo-langage est de pouvoir reformuler les éléments d'un modèle source, qui ne respectent pas le modèle de la métabase, en utilisant exclusivement le vocabulaire défini dans le modèle conceptuel de la métabase (qui se trouve en annexe 5.1.2).

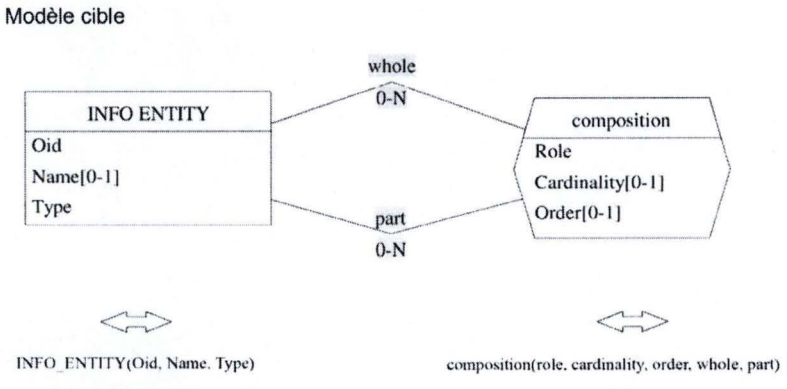


Figure 21 Sous-ensemble du modèle de la métabase

La figure ci-dessus montre une partie du modèle de la métabase représenté graphiquement et en pseudo-code.

Imaginons que l'on veuille définir une règle pour transformer une entité *bd* appartenant au type d'entité « Base de donnée », en une instance *bd_ent,* de type « Info Unit » ; on écrira :

```
Base_de_donnee bd = Base_de_donnee(nom) ;
```

⇔

INFO_ENTITY bd_ent = INFO_ENTITY(bd_ent.nom, bd_ent.nom, 'Base de donnée') ;

La partie au dessus de l'équivalence fait référence à l'entité que l'on souhaite transformer ; la partie en dessous, définit ce que cette entité devient après transformation. Ainsi, le *nom* de *bd* sera affecté à l'*oid* de *bd_ent* ; le *nom* de *bd* sera aussi affecté à l'attribut *nom* de *bd_ent* ; et le *type* de *bd_ent* prendra une valeur : "Base de donnée".

Si l'on veut définir une règle pour transformer une instance du TA « structure » du modèle d'origine en instance du TA « composition » dans le modèle destination, on écrira :

structure(bd, table) ;

⇔

composition('structure', _ , bd_ent, table_ent) ;

où bd_ent référence l'entité d'information représentant bd ; table_ent référence l'entité d'information représentant table

Dans ce cas, on n'oubliera pas de préciser à quoi font référence *bd* et *table* ; dans ce cas, *bd* fait référence à une entité « Base de donnée » et *table* à une entité de type « Table ». La deuxième partie de la règle signifie que l'attribut *type* du TA « composition » prendra une valeur fixe : "structure" ; les attributs *cardinality* et *order* auront une valeur nulle ; le rôle *whole* fera référence à l'« Info Unit » *bd_ent* qui issu de la transformation de *bd,* représente cette entité dans le modèle de la métabase ; ...

2.4. Représentation des documents non structurés

La représentation des documents non structurés est le cas le plus simple. Il s'agit de document dont le contenu est composé d'une suite de mots et éventuellement de données multimédia (image, son, vidéo).

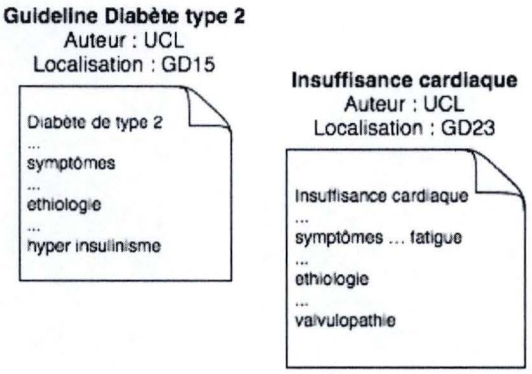


Figure 22 Exemples de documents textes

Le contenu peut être très volumineux surtout s'il contient des données multimédias ; c'est pour cela que l'on ne stocke pas l'entièreté du contenu d'un document, mais uniquement certains termes, jugés pertinents pour représenter le contenu du document, qui constitueront l'index (voir chapitre 1.1.2).

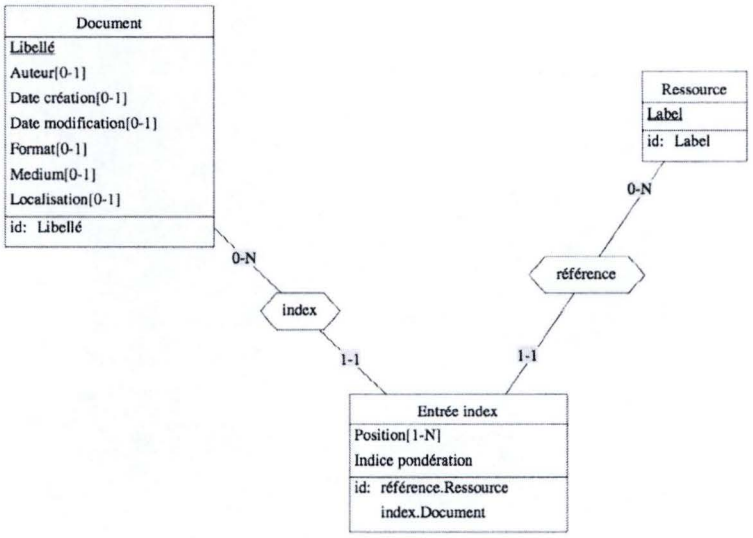


Figure 23 Modèle de description d'un document dans la métabase

Un **document** est constitué de méta-données comme le nom des **auteurs**, la **date de création**, la **date de modification**, le **format**, ... On retiendra aussi toute information permettant d'accéder au document comme son chemin d'accès ou son adresse (localisation). Le contenu du document est représenté par son index. Cet index est composé d'un ensemble d'**entrées**. Chaque **entrée de l'index** renvoie aux positions dans le document original où l'on retrouve le terme ainsi qu'un éventuel indice de pondération. L'indice de pondération est utilisé lors du calcul de la pertinence si tous les concepts n'ont pas la même importance, par exemple le nombre de fois où le

concept se retrouve dans le document. Il n'y a qu'une **entrée dans l'index** par **document** et par **ressource**. Une **ressource** représente le terme indexé ; il peut s'agir d'un mot ou d'un concept dans une ontologie comme nous le verrons au chapitre 2.8.2.

Transformations

Nous allons voir comment la description d'un document respectant le modèle de la Figure 23 peut être décrit dans un formalisme respectant le modèle de la métabase.

Le langage utilisé pour définir les transformations est détaillé dans la section 2.3.1. Nous reprenons ici les notations utilisées pour décrire les structures du modèle la métabase.

```
INFO_ENTITY(oid, name, type, typeParameters, medium, location) ;
PROPERTY(name, type, ?value) ;
object_attribute([INFO_ENTITY) entity, (PROPERTY) attibute) ;
info-relation(type, (INFO_ENTITY) merber1, (INFO_ENTITY) merber1) ;
composition(role, cardinality, order, (INFO_ENTITY) whole, (INFO_ENTITY) part) ;
```

La première règle spécifie comment transformer un **document** et ses métadonnées. Un **document** est représenté par une « INFO_ENTITY » de type 'document'. Chacune de ses métadonnées devient une « PROPERTY » attachée à l' « INFO_ENTITY » représentant le **document**.

```
Document doc = Document(libelle, auteur, date_creation, date_modif, format, medium, localisation) ;
⇔
INFO_ENTITY doc_ent = INFO_ENTITY(doc.libelle, doc.libelle, 'Document Plat', doc.format, doc.medium,
doc.localisation) ;
PROPERTY auteur_doc = PROPERTY('auteur', doc.auteur, _) ;
object_attribute(document, auteur_doc) ;
PROPERTY date_creation_doc = PROPERTY('date creation, doc.date_creation, _) ;
object_attribute(document, date_creation) ;
PROPERTY date_modification_doc = PROPERTY('date modification, doc.date_modif, _) ;
object_attribute(document, date_modification) ;
```

Chaque **entrée de l'index** est transformée en une « INFO_ENTITY ». Les **positions** du terme dans le document seront stockées dans une entité de type « PROPERTY ». On aura alors autant d'instances « PROPERTY » que d'occurrence de ce terme dans le document. Il s'agit de la modélisation la plus évidente, la plus simple et la plus évolutive. Cependant, cette solution peut s'avérer couteuse en espace de stockage et en temps d'accès.

```
Entree_index entree_index = Entree_indexe(positions[1-N], indice_ponderation) ;
⇔
```

INFO_ENTITY entree_index = INFO_ENTITY(entree_index.document.libelle · entree_index.ressource->label, _, 'Entrée index', indice_ponderation, _, _) ;

∀ pos ∈ positions : PROPERTY pos_t_i = PROPERTY(_, 'position', pos) ;
object_attribute(entree_index, position_t_i) ;

Une autre solution pour retenir cette liste consiste à concaténer les valeurs. En utilisant une « Info-Entity » plutôt qu'une « Info-Unit » pour représenter le terme indexé, le résultat de la concaténation peut être stocké dans l'attribut « valeur ». Cette solution corrige les inconvénients de la première solution, mais a pour inconvénients d'être moins expressive et rends la gestion de la liste plus complexe.

Une référence entre une **entrée de l'index** *t_i* et une **ressource** *ressource* qui représentent un terme est enregistrée dans une **info-relation** de type « référence ».

reference(t_i, ressource);
<=>
info-relation('référence', terme_index_ent, ressource_ent);
où terme_index_ent référence l'entité d'information représentant t_i et ressource_ent référece l'entité d'information représentant ressource.

Un lien entre une **entrée de l'index** *index* et un **document** *document* est encodé dans une **info-relation** de type « index ».

index(document, entree_index);
⇔
info-relation('index', document_ent, entree_index_ent);
où document_ent référence l'entité d'information représentant document et entree_index_ent référence l'entité d'information représentant entree_index.

Le schéma ci-dessous représente le résultat de l'indexation du document « Guideline Diabète type 2 » (Figure 22) en utilisant les règles définies dans cette section. Le document est indexé selon les termes « Diabète de type 2 » qui est le 108ème caractère dans le texte et « Symptômes » (545ème caractère).

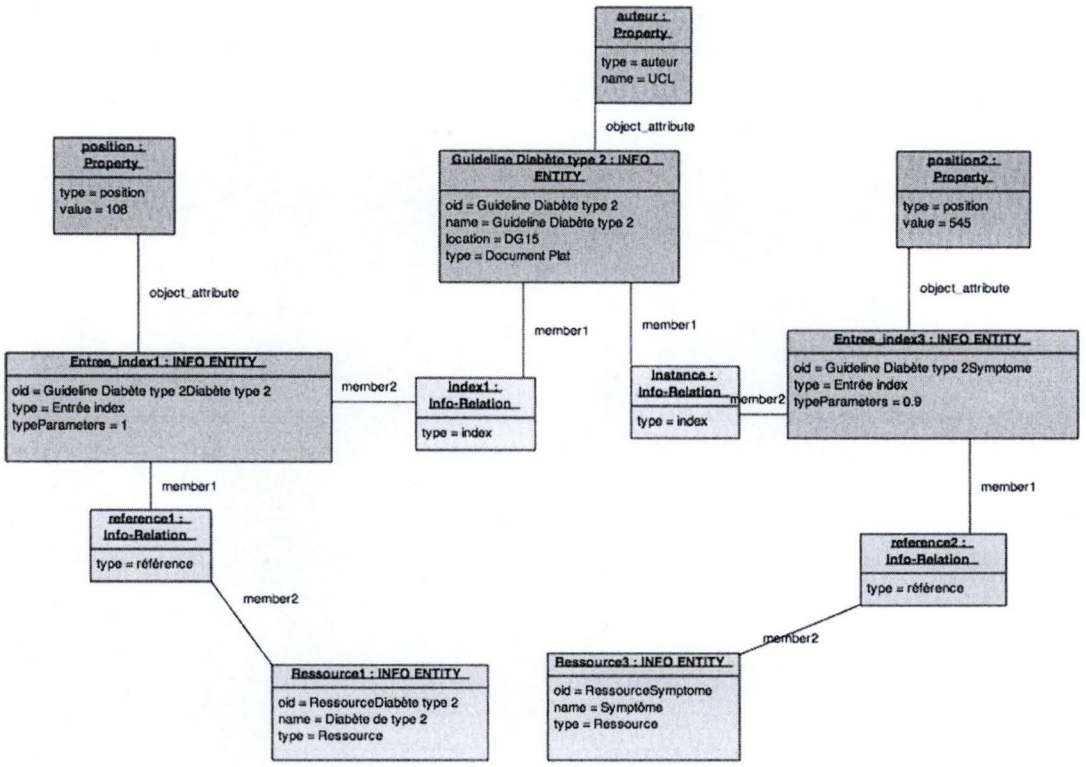


Figure 24 Exemple de document encodé en utilisant le vocabulaire de la métabase

Il est aussi possible d'enregistrer l'entièreté du contenu du document, sous la forme d'une suite de byte, en utilisant le type d'entité « Info-Unit ».

2.5. Représentation des documents semi-structurés

Nous allons détailler le cas du langage de balisage le plus répandu : XML. Un document XML utilise une structure arborescente où chaque nœud renferme du contenu encadré de balises. Le contenu peut contenir d'autres nœuds, du texte, des images, ... La structure d'un document XML est définie par une DTD ou un XML Schéma. On décrira la façon de stocker la description d'une instance de document XML après avoir abordé le cas structure.

2.5.1. La structure

XML ne permet pas définir de modèles pour exprimer des contraintes sur la forme de ses instances. Il n'est donc pas possible de connaître la structure d'un tel document a priori. Afin de combler ce manque, des langages de définition de schémas existent. Un schéma permet de définir la syntaxe d'une série de documents XML. Un document XML est valide par rapport à un schéma s'il respecte les contraintes qui y sont définies. Les langages de définition de schéma les plus populaires sont DTD (Document Type

Définition) et XSD (XML Schema Language). Les DTD sont simples d'utilisation et faciles à comprendre, mais ne permettent pas d'exprimer certaines contraintes comme les domaines de valeurs. Le langage définit un mécanisme d'intégrités référentielles malheureusement très limité. A cause des limites des DTD, le W3C a proposé le langage XSD pour les remplacer. Le langage XSD est plus complexe, mais beaucoup plus puissant. Ce langage utilise une notation XML alors que les DTD utilisent une syntaxe qui leur est propre.

```
<Pathologies>
   <Pathologie id=1>
      <nom>
      ... pneumonie ...
      </nom>
      <etiologie>
      ... infections bactériennes
... virus ...
      </etiologie>
   </Pathologie>
   <Pathologie id=2>
      ...
   </Pathologie>
</Pathologies>
```

```
<?xml version="1.1">!
<!DOCTYPE pathologies
"guideline.dtd">!

<!ELEMENT pathologies
(pathologie*)>!
<!ELEMENT pathologie (nom,
etiologie)>!
<!ELEMENT nom (#PCDATA)>!
<!ELEMENT etiologie (#PCDATA)>!
```

Figure 25 Extrait de document XML (gauche) et sa DTD (droite)

Nous aborderons seulement le cas des DTD, plus lisibles et plus répandus. Une DTD est composée d'un ensemble de déclarations : d'éléments, de liste d'attributs et d'entités. Nous ne traiterons pas le cas des entités qui permettent de définir des raccourcis.

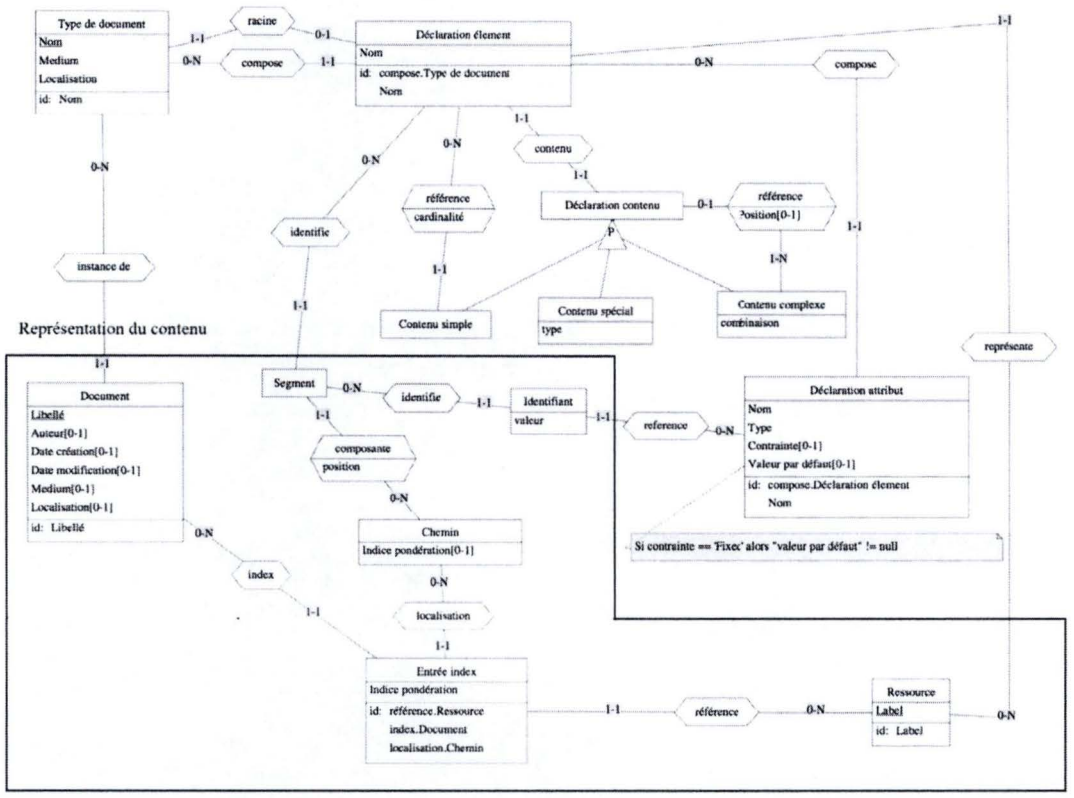


Figure 26 Schéma DTD et représentation des index XML

La racine d'une DTD déclare le nom de l'élément racine et contient un ensemble de déclarations d'éléments.

Une déclaration d'**élément** permet de définir son nom et son contenu. Le **contenu** d'un élément (Contenu élément) est simple (Contenu simple), spécial (Contenu spécial) ou complexe (Contenu complexe). Un **contenu simple** est caractérisé sur une contrainte de cardinalité (cardinalité) portant sur l'**élément** auquel il fait référence. Les contraintes de cardinalité possibles sont facultatif (?), au moins un (+), zéro ou plusieurs (*). Un **contenu spécial** est caractérisé par le type de contenu qui peut être une chaine de caractère (#PCDATA), n'importe quelle valeur (ANY) ou aucune valeur (EMPTY). Un **contenu complexe** permet de combiner différents types de contenu. Il est caractérisé par la façon de combiner les déclarations de contenu auquel il fait référence. Les types de combinaison possible sont la concaténation (,) et l'union (|).

Une **déclaration d'attribut** est caractérisée par un **nom**, un **type** et une éventuelle **contrainte** sur la valeur et une éventuelle **valeur par défaut**. Le type de l'attribut peut être n'importe quel caractère (CDATA), une énumération, un identifiant (ID) ou une référence à un identifiant (IDREF). Un attribut de type identifiant permet d'identifier un élément dans un document XML. Il doit être unique à tout le document et pas seulement entre les mêmes types d'éléments comme avec les bases de données

relationnelles. Il peut y avoir au maximum un attribut de type identifiant par élément. Un attribut de type IDREF contient un ensemble, qui peut être vide, de valeur d'identifiant. Chaque valeur doit correspondre à l'identifiant d'un élément présent dans le document. La **contrainte** sur la valeur permet de spécifier le caractère obligatoire (#REQUIRED), facultatif (#IMPLIED) ou (#FIXE) de la valeur de cet attribut. Une **valeur par défaut** doit être fournie si l'attribut est contraint à prendre une valeur fixe.

Transformations

Nous allons voir comment une DTD respectant le modèle représenté à la figure Figure 26 peut être décrite dans un formalisme respectant le modèle de la métabase.

Le langage utilisé pour définir les transformations est détaillé dans la section 2.3.1. Nous reprenons ici les notations utilisées pour décrire les structures du modèle la métabase.

```
INFO_ENTITY(oid, name, type, typeParameters, medium, location) ;
PROPERTY(name, type, ?value) ;
object_attribute((INFO_ENTITY) entity, (PROPERTY) attibute) ;
info-relation(type, (INFO_ENTITY) merber1, (INFO_ENTITY) merber1) ;
composition(role, cardinality, order, (INFO_ENTITY) whole, (INFO_ENTITY) part) ;
```

Un **type de document** est transformé en une **entité d'information**. Le nom du document sert d'identifiant (oid) au type document.

```
Type_de_document doctype = Type_de_document(nom, medium, localisation) ;
⇔
INFO_ENTITY doctype_ent = INFO_ENTITY(doctype.nom, doctype.nom, 'DTD', _, medium, localisation) ;
```

Chaque **déclaration d'élément** est transformée en entité d'information. La concaténation du nom du document et du nom de l'élément sert d'identifiant aux déclarations d'éléments.

```
Declaration_element elementdecl = Declaration_element(nom)
⇔
INFO_ENTITY elementdecl_ent = INFO_ENTITY(doctype.nom · element.nom, element.nom, 'Déclaration élément', _ _
_) ;
```

On retient le lien entre un **type de document** *doctype* et la déclaration d'élément racine *elementdecl* en utilisant une « info-relation » auquel on donne le type « racine ».

```
racine(doctype, elementdecl)
⇔
info-relation('racine', doctype_ent, elementdecl_ent) ;
```

où doctype_ent référence l'entité d'information représentant doctype ; elementdecl_ent référence l'entité d'information représentant elementdecl.

On lie chaque déclaration d'élément *elementdecl* au type de document *doctype* qui les agrège en utilisant le type d'association **composition.**

compose(doctype, elementdecl)

⇔

composition('compose', _, _, *doctype*_ent, elementdecl_ent)

où doctype_ent référence l'entité d'information représentant doctype ; elementdecl_ent référence l'entité d'information représentant elementdecl.

On utilise une **entité d'information** pour représenter un **contenu simple** et une relation de **composition** pour le lier à la **déclaration d'élément** *elementdecl* auquel il fait référence. Afin d'obtenir un identifiant unique pour l'**oid** de l'**entité d'information**, on concatène l'**oid** de la déclaration de l'élément et un numéro unique a tous les **contenus d'élément** attaché à cette **déclaration d'élément**.

Contenu_simple contenu_simple = Contenu_simple() ;

reference(cardinalité, contenu_simple, elementdecl) ;

⇔

INFO_ENTITY contenu_simple_ent = INFO_ENTITY(elementdecl.oid · uniqueNumber, _, 'Contenu simple', _, _, _) ;

composition('référence', cardinalité, _, *elementdecl*_ent, contenu_simple_ent) ;

où elementdecl_ent référence l'entité d'information représentant elementdecl.

Un **contenu spécial** est transformé en une **unité d'information** qui a pour valeur le type de contenu.

Contenu_special contenu_special = Contenu_special(type) ;

⇔

INFO_UNIT(elementdecl.oid · uniqueNumber, _, 'Contenu spécial', _, _, _, type) ;

Un **contenu complexe** est transformé en une **unité d'information** qui a pour valeur le type de combinaison des **déclarations de contenu** auquel il fait référence. Chaque référence à une déclaration de contenu *contenu* est enregistrée à l'aide du type d'association **composition**.

Contenu_complexe contenu_complexe = Contenu_complexe(combinaison)

∀ contenu_complexe.reference : reference(contenu_complexe, contenu, position)

⇔

INFO_UNIT contenucomplexe_ent = INFO_UNIT(elementdecl.oid · uniqueNumber, _, 'Contenu complexe', _, _, _, combinaison) ;

∀ contenu_complexe.reference : composition('référence', _, position, contenucomplexe_ent, contenu_ent) ;

où contenu_ent référence l'entité d'information représentant contenu.

Pour chaque **déclaration d'élément** *elementdecl*, on enregistre le lien vers la **déclaration de son contenu** *contenu*.

contenu(elementdecl, contenu)

⇔

info-relation('contenu', elementdecl_ent, contenu_ent) ;

où contenu_ent référence l'entité d'information représentant contenu ; elementdecl_ent référence l'entité d'information représentant elementdecl.

Chaque **déclaration d'attribut**, attachée à une **déclaration d'élément** est transformée en **entité d'information** auquel on attache un ensemble de **propriétés** contenant les valeurs de ses attributs (type, contrainte, valeur par défaut). Le lien entre la déclaration de l'élément *elementdecl* et celle de l'attribut est enregistré dans une relation de **composition**.

Declaration_attribut attributdecl = Declaration_attribut(nom, type, contrainte, valeur_par_defaut) ;

compose(elementdecl, attributdecl) ;

⇔

INFO_ENTITY attributdecl_ent = INFO_ENTITY(elementdecl_ent.oid · attributdecl.nom, attributdecl.nom, 'Déclaration attribut', _, _, _) ;

PROPERTY type_attributdecl_ent = PROPERTY('type', attributdecl.type, _) ;

object_attribute(attributdecl_ent, type_attributdecl) ;

PROPERTY contrainte_attributdecl_ent = PROPERTY('contrainte', attributdecl.contrainte, _) ;

object_attribute(attributdecl_ent, contrainte_attributdecl) ;

PROPERTY valeur_par_defaut_attributdecl = PROPERTY('valeur par défaut', attributdecl.valeur_par_defaut, _) ;

object_attribute(attributdecl_ent, valeur_par_defaut_attributdecl) ;

composition('compose', _, _, elementdecl_ent, attributdecl_ent) ;

où elementdecl_ent référence l'entité d'information représentant elementdecl.

2.5.2. Les instances

Un « document » (représenté dans l'encadré Figure 26) est caractérisé par un « libellé » et une série d'informations facultatives comme le nom de son « auteur », sa « date de création », la date de la dernière modification, son support (medium) et une adresse permettant d'y accéder (localisation). Un document est une instance d'un « type de document » (via le TA : instance de) décrivant sa structure.

Terme	Chemin
pneumonie	/Pathologies/Pathologie[@id='1']/nom
infections	/Pathologies/Pathologie[@id='1']/etiologie

Figure 27 Exemple d'index pour un document XML

Le contenu d'un document est représenté par un index composé d'**entrées**. Une **entrée de l'index** référence une **ressource** et peut être localisée (TA : localisation) à l'aide d'un **chemin** identifiant un nœud dans le document. Une **ressource** représente le terme indexé ; il peut s'agir d'un mot ou d'un concept dans une ontologie comme nous le verrons au chapitre 2.8.2. Un **chemin** (Ex. : "/Pathologies/Pathologie[@id='1']/nom") est composé de **segments** et peut avoir un **indice de pondération**. Un **segment** se trouve à une certaine **position** dans le **chemin**. Un **segment** (un nœud) est composé du

nom d'un élément (Ex. : "Pathologie") et éventuellement des valeurs de ses attributs permettant d'identifier le nœud de manière unique (Ex. : "id="1").

On a choisi de représenter la syntaxe du chemin permettant d'identifier un élément XML. On pourrait aussi l'enregistrer dans une simple chaine de caractère ; on perdrait alors en expressivité et le chemin pourrait être composé d'éléments inexistants, mais on gagnerait en performance.

Transformations

Nous allons voir comment la description d'un document XML respectant le modèle représenté à la figure Figure 26 (encadré en noir) peut être décrite dans un formalisme respectant le modèle de la métabase.

Le langage utilisé pour définir les transformations est détaillé dans la section 2.3.1. Nous reprenons ici les notations utilisées pour décrire les structures du modèle la métabase.

```
INFO_ENTITY(oid, name, type, typeParameters, medium, location) ;
PROPERTY(name, type, ?value) ;
object_attribute((INFO_ENTITY) entity, (PROPERTY) attibute) ;
info-relation(type, (INFO_ENTITY) merber1, (INFO_ENTITY) merber1) ;
composition(role, cardinality, order, (INFO_ENTITY) whole, (INFO_ENTITY) part) ;
```

Un **document** est transformé en une **entité d'information.** Chacune de ses métadonnées devient une **propriété.**

```
Document document = Document(libelle, auteur, date_creation, date_modif, medium, location) ;
⇔
INFO_ENTITY doc_ent = INFO_ENTITY(doc.libelle, doc.libelle, 'Document XML', _, doc.medium, doc.location) ;
PROPERTY auteur_doc = PROPERTY('auteur', document.auteur, _) ;
object_attribute(doc_ent, auteur_doc) ;
PROPERTY date_creation_doc = PROPERTY('date creation, document.date_creation, _) ;
object_attribute(doc_ent, date_creation) ;
PROPERTY date_modification _doc = PROPERTY('date modification, document.date_modif, _) ;
object_attribute(doc_ent, date_modification) ;
```

Un **chemin** est transformé en **unité d'information** dont la valeur est son **indice de pondération.**

```
Chemin chemin = Chemin(indice_ponderation) ;
⇔
INFO_UNIT chemin_ent = INFO_UNIT(_ _ 'Chemin accès nœud XML, _ _ , chemin.indiceponderation) ;
```

Un **segment** est enregistré dans une **entité d'information**. Le lien vers l'**élément** *declaration_element* est enregistré dans une « **info-relation** ». Le lien vers le **chemin** *chemin* est enregistré dans une relation de **composition**.

Segment segment = Segment() ;

identifie(segment, declaration_element) ;

Composante composante = composante(segment, chemin, position) ;

⇔

INFO_ENTITY segment_ent = INFO_ENTITY(␣ ␣ 'Valeur identifiant un segment', ␣ ␣ ␣ ␣) ;

info-relation('identifie', segment_ent, attribut_ent) ;

composition('agrège', ␣ composante.position, chemin_ent, segment_ent) ;

où declaration_element_ent référence l'entité d'information représentant declaration_element; chemin_ent référence l'entité d'information représentant le chemin.

Une valeur identifiant un **segment** est représentée en **unité d'information**. La référence vers l'**attribut** *attribut_ent* auquel elle s'applique et le **segment** *segment* qu'elle permet d'identifier sont enregistrés dans une « **info-relation** ».

Identifiant identifiant = Identifiant(valeur) ;

reference(identifiant, attribut) ;

identifie(identifiant, segment) ;

⇔

INFO_UNIT identifiant_ent = INFO_UNIT(␣ ␣ 'Valeur identifiant segment', ␣ ␣ ␣ ␣ identifiant.valeur) ;

info-relation('reference', identifiant_ent, attribut_ent) ;

info-relation('identifie', identifiant_ent, segment_ent) ;

où attribut_ent référence l'entité d'information représentant l'attribut ; segment_ent référence l'entité d'information représentant le segment.

Chaque **entrée de l'index** est transformée en **unité d'information** dont la valeur est l'**indice de pondération**. Les liens avec le **document** *document*, le **chemin** *chemin* et la **ressource** *ressource* sont enregistrés dans une **info-relation**.

Entree_index entree_index = Entree_index(indice_ponderation) ;

index(entree_index, document) ;

reference(entree_index, ressource) ;

localisation(entree_index, chemin) ;

⇔

INFO_ENTITY terme_indexe_ent = INFO_ENTITY(doc_ent.oid · chemin_ent.oid · ressource_ent.oid, ␣ 'Terme indexé', ␣ ␣ ␣ ␣ entreeindex.indiceponderation) ;

info-relation('index', entree_index_ent, doc_ent) ;

info-relation('référence', entree_index_ent, ressource_ent) ;

info-relation('localisation', entree_index_ent, chemin_ent) ;

où doc_ent référence l'entité d'information représentant le document ; ressource_ent référence l'entité d'information représentant la ressource et chemin_ent référence l'entité d'information représentant le chemin.

Le schéma ci-dessous représente la DTD et une partie du document XML de la Figure 25 exprimé dans le modèle de la méta-base. En (1), on retrouve la racine de la DTD. En (2) la racine du document XML. En (3), le chemin du nœud « /Pathologies/Pathologie » qui index les termes « Infection bactérienne » et « virus ».

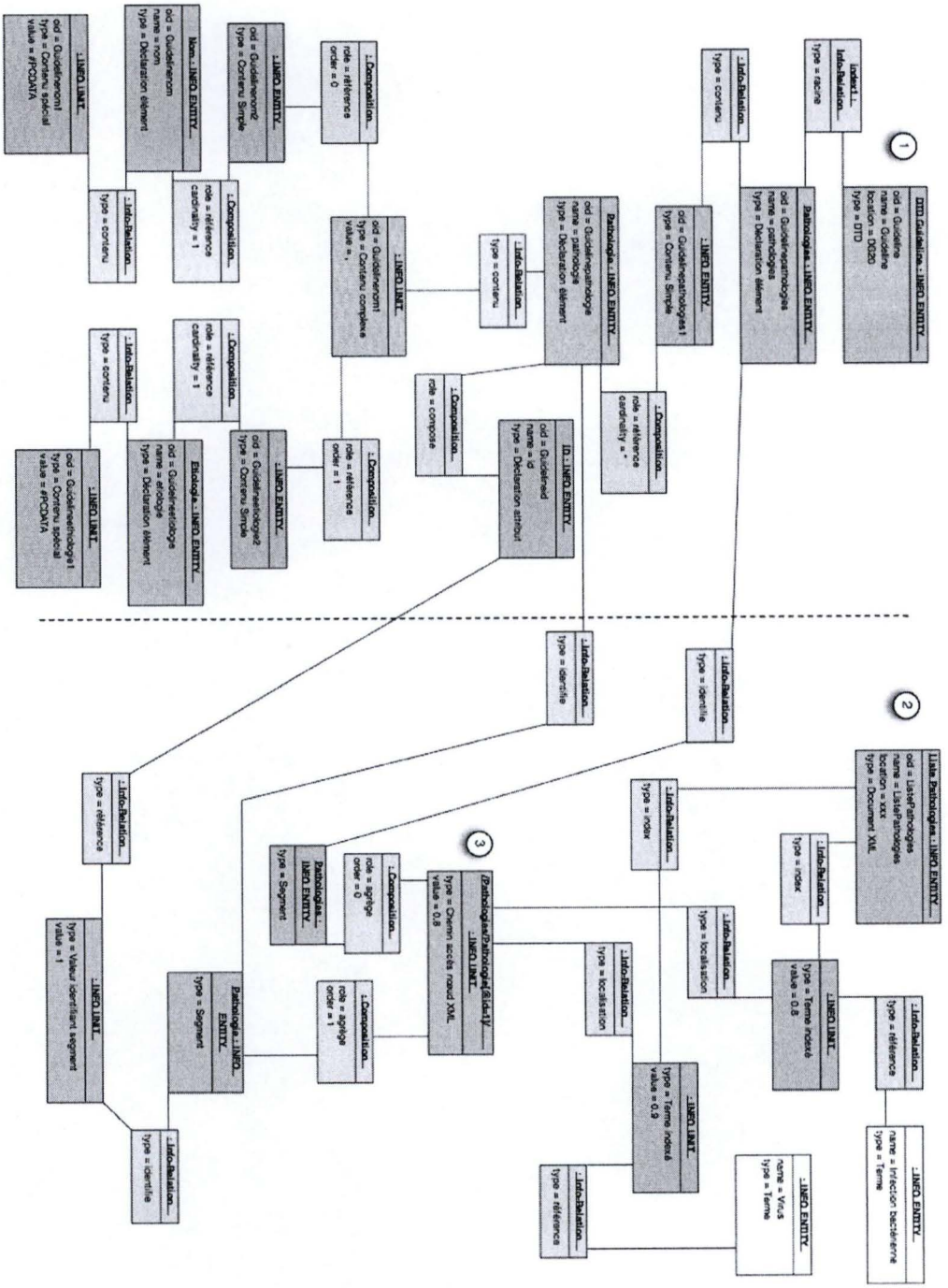


Figure 28 DTD et Index de document XML exprimé en utilisant le vocabulaire de la métabase

2.6. Représentation de bases de données

Pour la description des bases de données, on supposera que son concepteur a suivi la méthode de développement décrit dans (Hainaut, 2009). Cette méthode considère la conception d'une base de données comme un enchainement de processus. Chacun prenant un document en entrée pour le transformer en un nouveau document comme le montre la figure ci-dessous qui illustre leur enchainement :

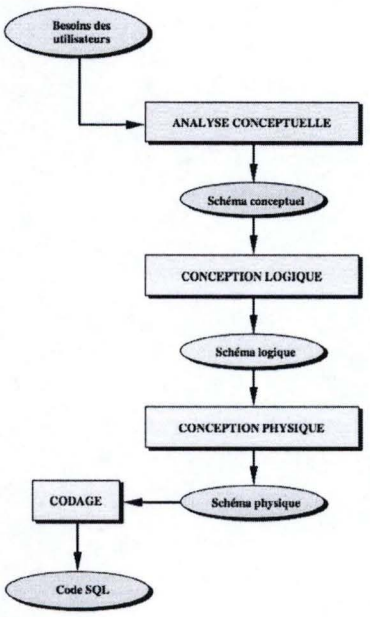


Figure 29 Processus développement BD - (Hainaut, 2003)

L'*analyse conceptuelle* consiste à traduire de manière formelle les besoins des utilisateurs sous la forme d'un schéma conceptuel. Lors de la *conception logique*, le schéma conceptuel sera transformé en un schéma logique en adaptant ses composantes à une famille de SGBD (Relationnel, Orienté objet, ...). Lors de la *conception physique*, ce schéma sera optimisé pour un SGBD particulier (Oracle, MySQL, Access, ...). Finalement, le schéma sera traduit dans un langage de description de donnée supporté par le SGBD, par exemple en SQL-DDL. Pour les bases de données qui n'ont pas suivi ce processus, il est possible de reconstruire les schémas logique et physique par rétro-ingénierie (Hainaut, 2009, p. 22).

À l'exception des besoins utilisateurs, les processus de transformations de schémas sont formalisables sous forme d'un ensemble de règles. Cela permet d'automatiser la génération d'une base de données à partir du schéma conceptuel tout en gardant la trace des liens entre les différents schémas, comme nous le verrons dans la section 2.8.1.

Seuls les schémas conceptuel et logique seront stockés dans la base de métadonnées. Nous avons écarté le schéma physique et le code DDL car les informations qu'ils apportent comme la présence d'index sont plutôt utiles à l'administrateur de la BD qui cherche à optimiser les performances qu'à nous qui ne sommes qu'utilisateurs.

Le schéma conceptuel est utile, car il formalise les besoins des utilisateurs en faisant abstraction des contraintes liées aux technologies utilisées pour l'implémentation. Les modèles conceptuels, plus expressifs que les modèles logiques, sont plus faciles à comprendre pour un humain.

Le schéma logique est indispensable, car il permet de savoir comment les données sont régies par le SGBD.

Nous allons d'abord voir comment représenter des schémas conceptuels dans la base de métadonnée ; pour ensuite nous intéresser au cas des schémas logiques.

2.6.1. Schéma conceptuel

Parmi les différents modèles conceptuels existants (UML, NIAM, ERA, ...), nous avons retenu le plus populaire : le modèle entités-relations-attributs étendu (ERA).

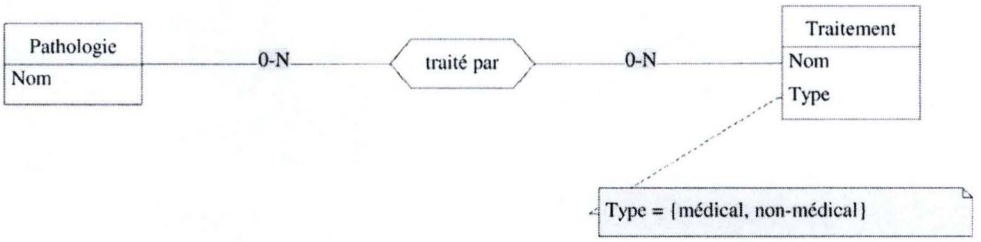


Figure 30 Exemple de schéma conceptuel – Version complète en annexe (section 5.1.3)

Un schéma conceptuel conforme au modèle ERA étendu représente les informations sous la forme d'un graphe. Ce graphe est composé de **types d'entités**, de **types d'associations**, d'**attributs** et d'**identifiants**.

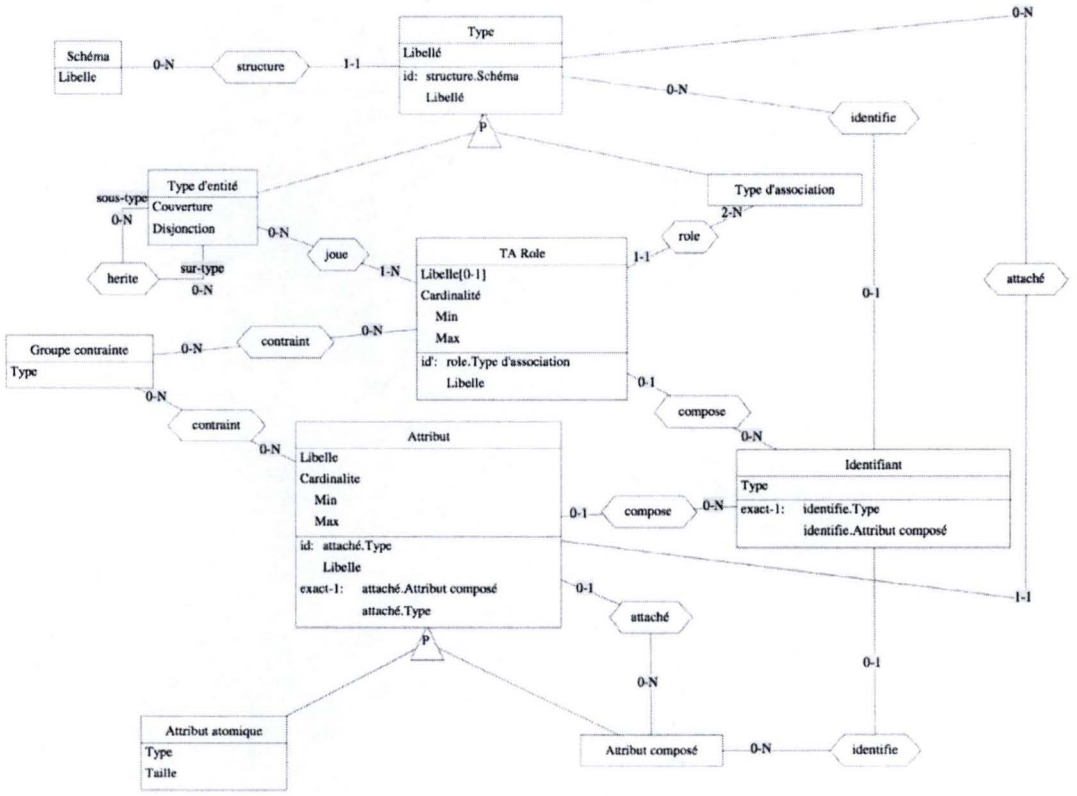


Figure 31 Modèle conceptuel

Un **type d'entité** peut hériter (TA **herite**) des caractéristiques d'un ou plusieurs autres **types d'entités**. Le **surtype** définit les relations qui existent entre ses populations et celles de ses éventuels **sous-types** par une contrainte de **disjonction** et une contrainte de **couverture**. Un **type d'association** comporte au moins deux **rôles**. Un **rôle** est joué par un ou plusieurs **types d'entités**. Il est soumis à une contrainte sur le nombre d'associations **minimales** (attribut **cardinalité min**) et **maximale** (attribut **cardinalité max**) entre lui et le(s) type(s) participant(s). Un **attribut** est atomique ou composé. Une intervalle [min-max] précise le nombre de valeurs **minimales** (attribut **cardinalité min**) et **maximales** (attribut **cardinalité max**) autorisées, cela permet de rendre l'attribut obligatoire ou facultatif, monovalué ou multivalué. Le domaine de valeur d'un attribut atomique est défini par un **type** de base (entier, chaine de caractère, ...) ayant une longueur (attribut **taille**) définie. Il est **attaché** à un **type d'entité**, un **type d'association** ou un **attribut composé**. Un **identifiant** est un ensemble **composé** d'**attributs** ou de **rôles** opposés. Il **identifie** un **type d'entités**, un **type d'associations** ou un **attribut composé**. Ces derniers peuvent posséder un identifiant primaire et plusieurs identifiants secondaires (enregistré dans l'attribut **type** du TE **Identifiant**). Une **contrainte de groupe** permet de définir un **type** de **contrainte** d'existence sur un ensemble d'**attributs** et/ou de **rôles**. Les **types** de **contrainte**s existantes sont la coexistence, l'exclusion, au moins un, exactement un et l'implication.

Certaines constructions conformes au modèle EA étendu n'ont pas été prises en compte ici. C'est le cas de contraintes comme celles portant sur le domaine de valeurs d'un attribut, les dépendances fonctionnelles, ... La possibilité de définir des types sur mesures par composition de type primitif, les « types définis par l'utilisateur », n'a pas non plus été prise en compte afin de ne pas surcharger ce document.

Transformation

Nous allons voir comment un schéma conceptuel respectant le modèle représenté à la figure Figure 31 peut être décrit dans un formalisme respectant le modèle de la métabase.

Le langage utilisé pour définir les transformations est détaillé dans la section 2.3.1. Nous reprenons ici les notations utilisées pour décrire les structures du modèle la métabase.

```
INFO_ENTITY(oid, name, type, typeParameters, medium, location) ;
PROPERTY(name, type, ?value) ;
object_attribute((INFO_ENTITY) entity, (PROPERTY) attibute) ;
info-relation(type, (INFO_ENTITY) merber1, (INFO_ENTITY) merber1) ;
composition(role, cardinality, order, (INFO_ENTITY) whole, (INFO_ENTITY) part) ;
```

Le **schéma** est représenté par une **entité d'information**.

```
Schema schema = Schema(libelle) ;
⇔
INFO_ENTITY schema_ent = INFO_ENTITY(libelle, libelle, 'Schéma conceptuel', _ _ _)
```

Les **types d'entités** contenus dans le **schéma** *schema* sont transformés en **entités d'informations**. Leurs attributs sont enregistrés sous forme de propriétés attachées à l'**entité d'information** représentant le type d'entité.

```
Type_d_entite type_entite = Type_d_entite(libelle, couverture, disjonction) ;
⇔
INFO_ENTITY type_entite_ent = INFO_ENTITY(schema_ent.oid · type_entite.libelle, type_entite.libelle, 'Type d'entité',
_ _ _) ;
PROPERTY couverture_prop = PROPERTY('couverture', type_entite.couverture, _) ;
object_attribute(type_entite_ent, couverture_prop ) ;
PROPERTY disjonction_prop = PROPERTY('disjonction', type_entite.disjonction, _) ;
object_attribute(type_entite_ent, disjonction_prop ) ;
où schema_ent référence l'entité d'information représentant schema.
```

Les relations **d'héritage** entre un **type d'entité** père *type_entite_pere* et un type d'entité fils *type_entite_fils* sont transformées en **info-relation** de type « hérite ».

herite(type_entite_pere, type_entite_fils) ;

⇔

info-relation('hérite', type_entite_pere_ent, type_entite_fils_ent) ;

où type_entite_pere_ent référence l'entité d'information représentant type_entite_pere ; type_entite_fils_ent référence l'entité d'information représentant type_entite_fils.

Chaque **type d'association** contenu dans le schéma *schema* et ses **rôles** sont transformés en **entités d'informations**. Les liens qui existent entre eux sont enregistrés dans des **info-relations** de type « rôle ».

Type_d_association ta = Type_d_association(libelle) ;
∀ ta.role : Ta_role role = Ta_role(libelle, cardinaliteMin, cardinaliteMax) ;
role(ta, role) ;

⇔

INFO_ENTITY ta_ent = INFO_ENTITY(schema_ent.oid · ta.libelle, ta.libelle, 'Type d'association', _ _ _) ;
∀ ta.role : INFO_ENTITY role_ent = INFO_ENTITY(ta.oid · role.libelle, role.libelle, 'TA rôle', _ _ _) ;
info-relation('rôle', ta_ent, role_ent) ;

où schema_ent référence l'entité d'information représentant schema.

Pour chaque **rôle**, on enregistre le lien vers chaque type d'entité *type_entite* qui **joue** ce rôle sous forme d'**info-relation** de type « joue ».

joue(role, type_entite) ;

⇔

info-relation('joue', role_ent, type_entite_ent) ;

où type_entite_ent référence l'entité d'information représentant type_entite.

Chaque **attribut** attaché à un type *type* est transformé en **entités d'informations**. Leurs caractéristiques sont enregistrées sous forme de propriétés attachées à l'**entité d'information** représentant l'attribut. Les attributs simples :

Attribut_atomique attribut = Attribut_atomique(libelle, cardinaliteMin, cardinaliteMax, type, taille) ;

⇔

INFO_ENTITY attribut_entite = INFO_ENTITY(type_ent.oid · attribut.libelle, attribut.libelle, 'Attribut atomique', _ _ _) ;
PROPERTY card_min_prop = PROPERTY('cardinalité minimum', attribut.cardinaliteMin, _) ;
object_attribute(attribut_entite, card_min_prop) ;
PROPERTY card_max_prop = PROPERTY('cardinalité maximum', attribut.cardinaliteMax, _) ;
object_attribute(attribut_entite, card_max_prop) ;
PROPERTY type_prop = PROPERTY('type', attribut.type, _) ;
object_attribute(attribut_entite, type_prop) ;
PROPERTY taille_prop = PROPERTY('taille', attribut.taille, _) ;
object_attribute(attribut_entite, taille_prop) ;
où type_entite_ent référence l'entité d'information représentant type.

Les attributs composés :

Attribut_compose attr_compose = Attribut_compose(libelle, cardinaliteMin, cardinaliteMax) ;

⇔

INFO_ENTITY attr_compose_ent = INFO_ENTITY(type_ent.oid · attr_compose.libelle, attr_compose.libelle, 'Attribut composé', _ _ _) ;

PROPERTY card_min_prop = PROPERTY('cardinalité minimum', attribut.cardinaliteMin, _) ;

object_attribute(attr_compose_ent, card_min_prop) ;

PROPERTY card_max_prop = PROPERTY('cardinalité maximum', attribut.cardinaliteMax, _) ;

object_attribute(attr_compose_ent, card_max_prop) ;

où type_entite_ent référence l'entité d'information représentant type.

Chaque relation entre un **type d'entité** ou **d'association** *type* et un **attribut** est encodée dans une relation de **composition**.

attache(type, attribut)

⇔

composition('attaché, _ _ type_ent, attribut_ent) ;

où type_ent référence l'entité d'information représentant type ; attribut_ent référence l'entité d'information représentant attribut.

On applique le même principe pour les relations entre un **attribut composé** *attribut_compose* et un **attribut composé** ou **atomique** *attribut*.

attache(attribut_compose, attribut)

⇔

composition('attaché, _ _ attribut_compose_ent, attribut_ent) ;

où attribut_compose_ent référence l'entité d'information représentant type ; attribut_compose référence l'entité d'information représentant attribut.

Chaque identifiant est transformé en une **unité d'information** dont la valeur est le **type** d'identifiant. Le lien vers le **type** *type* ou **l'attribut composé** *attribut_compose* identifié est représenté par une **info-relation** de type « identifiant ».

Identifiant identifiant = Identifiant(type) ;

identifiant(identidiant, type) || identifiant(identifiant, attribut_compose) ;

⇔

INFO_UNIT identifiant_ent = INFO_UNIT(_ _ 'Identifiant', _ _ _ identifiant.type) ;

info-relation('identifie', identifiant_ent, type_ent) || info-relation('identifie', identifiant_ent, attribut_compose_ent) ;

où type_ent référence l'entité d'information représentant type; attribut_compose_ent référence l'entité d'information représentant attribut_compose.

Chaque lien entre un **identifiant** et un **rôle** ou un **attribut** formant **l'identifiant** est représenté par une relation de **composition**. Avec un **rôle** :

compose(identifiant, role) ;

⇔

composition('compose'), _ _ identifiant_ent, role_ent) ;

où role_ent référence l'entité d'information représentant role; identifiant_ent référence l'entité d'information représentant identifiant.

Avec un **attribut** :

compose(identifiant, attribut) ;

⇔

composition('compose', _ _ identifiant_ent, attribut_ent) ;

où attribut_ent référence l'entité d'information représentant attribut; identifiant_ent référence l'entité d'information représentant identifiant.

Chaque contrainte est représentée par une unité d'information dont la valeur est le type de contrainte. Les liens avec les attributs et les rôles sont encodés dans des **info-relations**.

Groupe_contrainte contrainte = Groupe_contrainte(type) ;

∀contrainte.contraint.Attribut attribut : contraint(contrainte, attribut) ;

∀contrainte.contraint.TA_Role role : contraint(contrainte, role) ;

⇔

INFO_UNIT contrainte_ent = INFO_UNIT(_ _ 'Groupe contrainte', _ _ _ contrainte.type) ;

∀contrainte.contraint.Attribut attribut : info-relation('contraint', contrainte_ent, attribut_ent) ;

∀contrainte.contraint.TA_Role role : info-relation('contraint', role_ent, type_ent) ;

Chaque lien entre un **type d'association ou type d'entité** *type* et le **schéma** *schema* qui les contient est enregistré dans une relation de **composition**.

structure(schema, type)

⇔

composition('structure', _ _ schema_ent, type_ent) ;

où schema_ent référence l'entité d'information représentant schema; type_ent référence l'entité d'information représentant type.

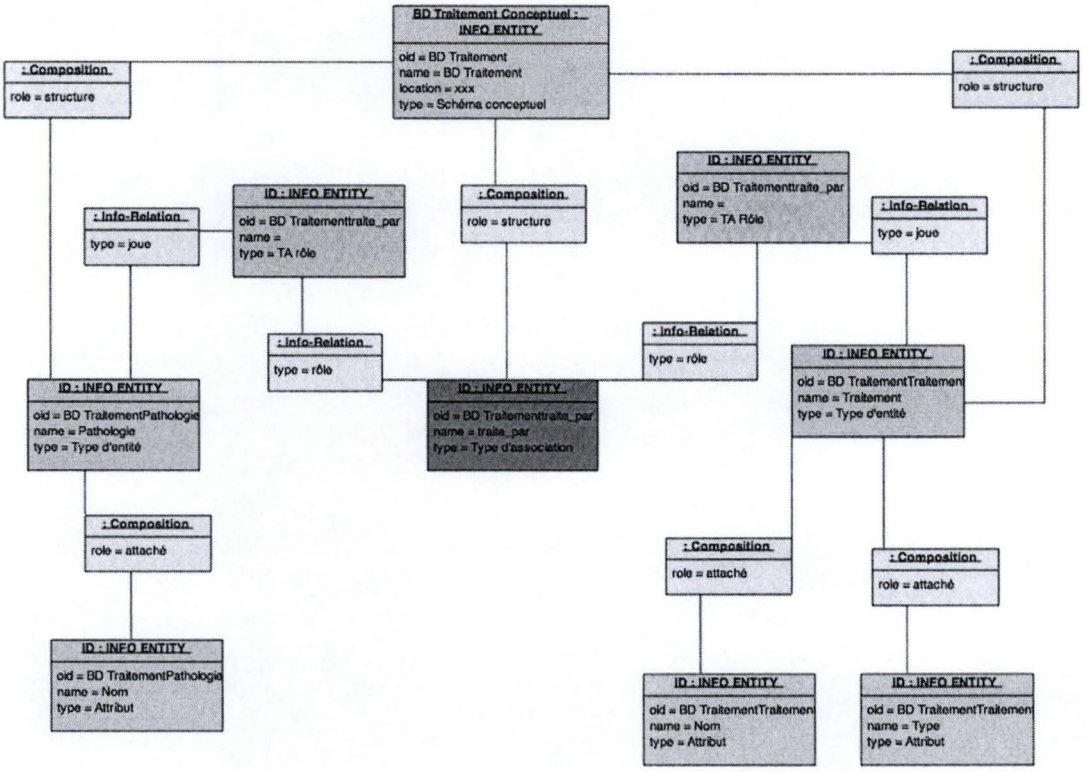


Figure 32 Schéma conceptuel exprimé en utilisant le vocabulaire de la métabase

Le schéma ci-dessus représente le schéma conceptuel de la Figure 30 exprimé dans le modèle de la métabase.

2.6.2. Schéma logique

Les bases de données relationnelles étant les plus répandues, notre choix s'est naturellement porté vers le modèle logique relationnel enrichi.

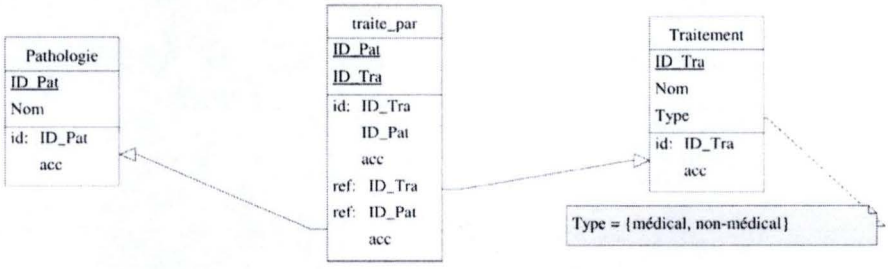


Figure 33 Schéma logique issu de la transformation du schéma conceptuel de la Figure 30

Un schéma est conforme au modèle logique relationnel s'il ne contient que des structures et des contraintes qui peuvent être directement traduite dans un SGBD

relationnel. Il est composé de tables, de colonnes et de contraintes comme les identifiants, les attributs de références, ...

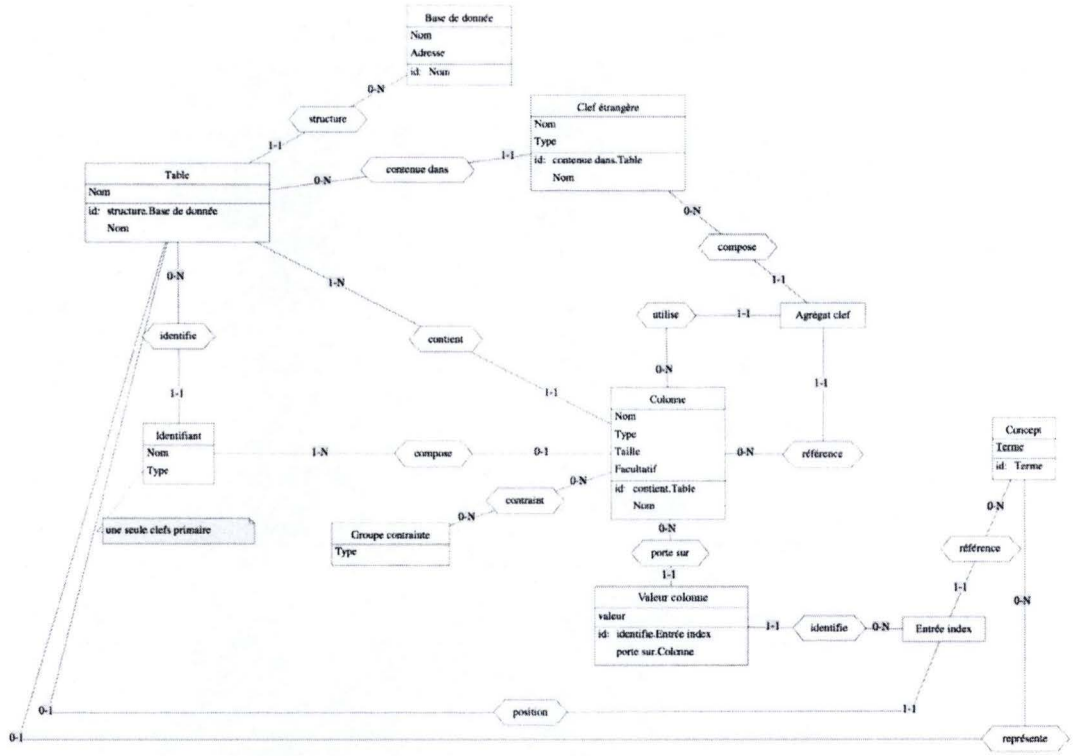


Figure 34 Modèle logique

Chaque **table** est constituée d'au moins une **colonne**. Une **colonne** est atomique et mono-valuée, son domaine de valeurs est défini par un type primitif. Elle est obligatoire ou facultative (null). Un **identifiant** est composé d'attributs. Une **table** peut avoir un **identifiant primaire** et plusieurs **identifiants secondaires** via le prédicat « unique ». Un ensemble de colonnes, appelé **clef étrangère**, peut référencer une des lignes d'une table. Une **clef étrangère** peut être totale. Un groupe de colonnes peut avoir à respecter une **contrainte d'existence** telle que la coexistence, l'exclusion, au moins un, exactement un et l'implication.

Une **entrée de l'index** référence un **concept** contenu dans une **table** à la ligne identifiable par un ensemble de **valeur de colonne**. Une **valeur colonne** permet de retenir la valeur d'une colonne.

Transformations

Nous allons voir comment un schéma logique respectant le modèle représenté à la figure Figure 34 peut être décrit dans un formalisme respectant le modèle de la métabase.

Le langage utilisé pour définir les transformations est détaillé dans la section 2.3.1. Nous reprenons ici les notations utilisées pour décrire les structures du modèle la métabase.

```
INFO_ENTITY(oid, name, type, typeParameters, medium, location) ;
PROPERTY(name, type, ?value) ;
object_attribute((INFO_ENTITY) entity, (PROPERTY) attibute) ;
info-relation(type, (INFO_ENTITY) merber1, (INFO_ENTITY) merber1) ;
composition(role, cardinality, order, (INFO_ENTITY) whole, (INFO_ENTITY) part) ;
```

Une **base de données** est représentée par une **entité d'information**.

```
Base_de_donnee bd = Base_de_donnee(nom, adresse)
⇔
INFO_ENTITY bd_entite = INFO_ENTITY(nom, nom, 'Schéma logique', _ _ adresse) ;
```

Chaque table est transformée en **entité d'information** auquel on associe à la représentation de la **base de données**.

```
Table table = Table(nom) ;
structure(bd, table) ;
⇔
INFO_ENTITY table_entite = INFO_ENTITY(bd_entite->oid · table.nom, table.nom, 'Table', _ _ _)
composition('structure', _ _ bd, table) ;
```

Les **colonnes** sont transformées en **entité d'information**. Le **type**, la **taille** et le caractère **facultatif** sont représentés sous forme de propriétés. On modélise le lien avec la table avec le type d'association **composition**.

```
Colonne colonne = Colonne(nom, type, taille, facultatif) ;
contient(table, colonne) ;
⇔
INFO_ENTITY col_entite = INFO_ENTITY(table.oid · colonne.nom, colonne.nom, 'Colonne', _ _ _) ;
PROPERTY type_prop = PROPERTY('type', colonne.type, _) ;
object_attribute(col_entite, type_prop) ;
PROPERTY taille_prop = PROPERTY('taille', colonne.taille, _) ;
object_attribute(col_entite, taille_prop) ;
PROPERTY facultatif_prop = PROPERTY('facultatif', colonne.facultatif, _) ;
object_attribute(col_entite, facultatif_prop) ;
composition('contient', _ _ table_entite, col_entite) ;
```

Les **identifiants** sont représenté sous forme d'**unité d'information** avec comme **valeur,** le type d'identifiant (primaire ou secondaire).

```
Identifiant identifiant = Identifiant(nom, type) ;
identifie(table, identifiant) ;
```

```
∀ identifiant.compose.colonne : compose(identifiant, colonne)
⇔
INFO_UNIT identifiant_entite = INFO_UNIT(table.oid · identifiant.nom, identifiant.nom, 'Identifiant', _ _ _
identifiant.type) ;
composition('identifie', _ _ table_entite, identifiant_entite) ;
composition('compose', _ _ identifiant_entite, col_entite) ;
```

Chaque **groupe de contrainte** est représenté sous forme d'**unité d'information** avec comme **valeur**, le type de contrainte.

```
Groupe_contrainte groupe = Groupe_contrainte(type) ;
∀ contraint.colonne : contraint(groupe, colonne)
⇔
INFO_UNIT groupe_entite = INFO_UNIT(colonne.oid · numeroUnique, _ 'Groupe contrainte, _ _ _ groupe.type) ;
composition('contraint', _ _ groupe_entite, col_entite) ;
```

Chaque **clé étrangère** est enregistrée dans une **unité d'information** ayant le type de clef pour **valeur**.

```
Clef_etrangere clef_etr = Clef_etrangere(nom, type) ;
contenue_dans(table, clef_etr) ;
⇔
INFO_UNIT clef_etr_entite = INFO_UNIT(table_entite.oid · clef_etr.nom, _ 'Clef étrangère', _ _ _ clef_etr.type) ;
composition('contient', _ _ table_entite, clef_etr_entite) ;
```

Un **agrégat de clef** est représenté par une entité d'information. Les liens avec l'entité représentant la clef étrangère, la colonne contenant pointant sur la colonne référencée et la colonne référencée sont enregistrés dans des info-relation.

```
Agregat_clef agregat_clef = Agregat_clef() ;
compose(agregat_clef, clef_etr) ;
reference(agregat_clef, colonne_reference) ;
utilise(agregat_clef, colonne_utilise) ;
⇔
INFO_ENTITY agregat_clef_entite = INFO_ENTITY(clef_etr_entite.oid · colonne_utilise.nom, _ 'Agrégat clef', _ _ _) ;
info-relation('composé de', clef_etr_entite, agregat_clef_entite) ;
info-relation('référence', agregat_clef_entite, colonne_reference_entite) ;
info-relation('utilise', agregat_clef_entite, colonne_utilise_entite) ;
```

Chaque **entrée de l'index** est transformée en **entité d'information**. Les liens avec la **table** *table* et le **concept** *concept* sont enregistrés dans une **info-relation**.

```
Entree_index entree_index = Entree_index(indice_ponderation) ;
index(entree_index, table) ;
reference(entree_index, concept) ;
⇔
INFO_UNIT terme_indexe_ent = INFO_UNIT(table_ent.oid · concept_ent.oid, _ 'Terme indexé', _ _ _) ;
info-relation('position', entree_index_ent, table_ent) ;
info-relation('référence', entree_index_ent, concept_ent) ;
```

où table_ent référence l'entité d'information représentant table; et concept_ent référence l'entité d'information représentant le concept.

Une **valeur de colonne** est transformée en **unité d'information**. Les liens avec la **colonne** *colonne* et l'**entrée dans l'index** *entree_index* sont enregistrés dans une **info-relation**.

Valeur_colonne valeur = Valeur_colonne(valeur) ;

porte_sur(valeur, colonne) ;

identifie(valeur, entree_index) ;

⇔

INFO_ENTITY valeur_ent = INFO_ENTITY(colonne_ent.oid · entree_index_ent.oid · colonne.valeur, _, 'Valeur colonne', _, _, _, valeur.valeur) ;

info-relation('porte sur, valeur_ent, colonne_ent) ;

info-relation('identifie, valeur_ent, entree_index_ent) ;

où colonne_ent référence l'entité d'information représentant colonne; et entree_index_ent référence l'entité d'information représentant le entree_index.

Le schéma ci-dessous représente le schéma logique de la Figure 33 exprimé dans le modèle de la métabase.

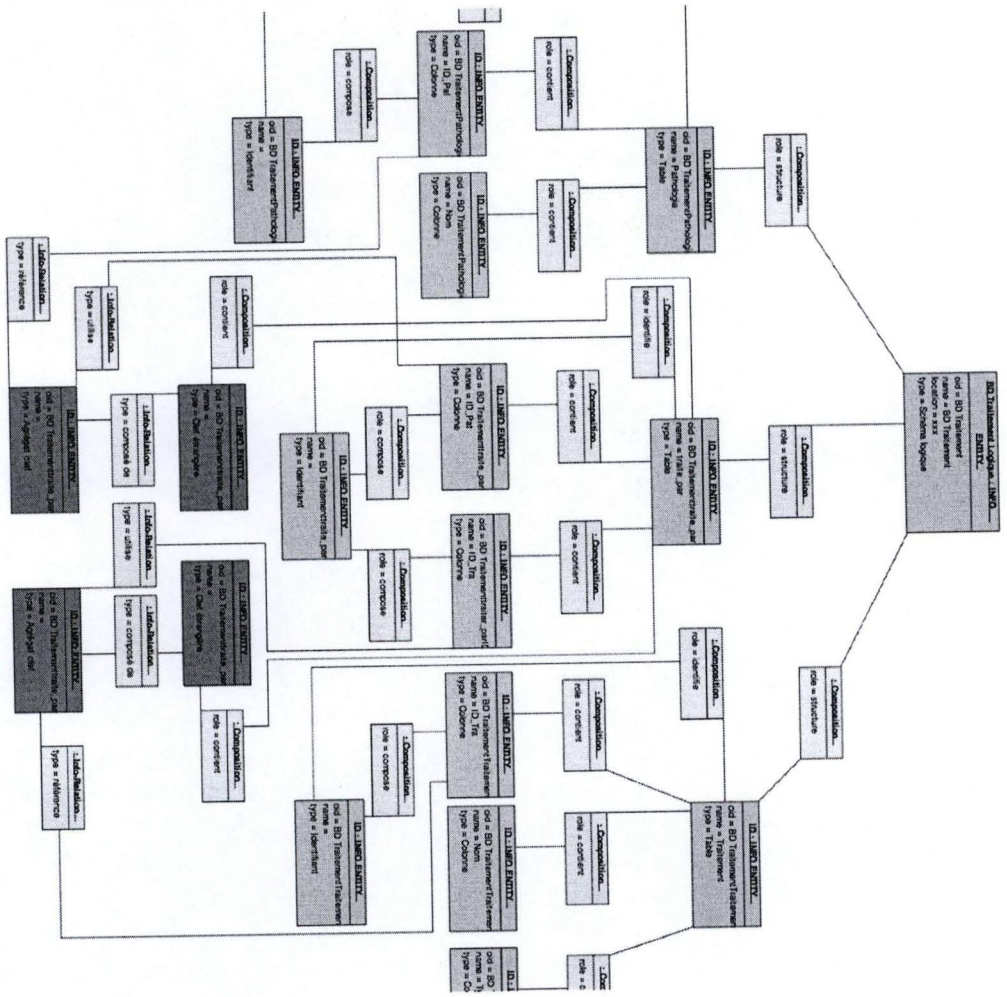


Figure 35 Schéma logique exprimé en utilisant le vocabulaire de la métabase

2.7. Représentation des ontologies

Les ontologies se classent dans la catégorie des données structurées. Nous avons choisi de leur consacrer un chapitre à part, car comme nous le verrons dans le chapitre 2.8.2, les ontologies vont nous permettre de donner du sens aux documents qu'ils soient structurés ou non. Contrairement aux autres représentations dont on ne stockait que la structure et une partie du contenu sous forme d'index, on stockera l'entièreté des ontologies dans la métabase. Cette différence de traitement vient du fait qu'une ontologie ne sépare pas les définitions de concepts de leurs instances. Nous avons choisi RDFS comme langage de départ pour l'importation d'ontologie dans la métabase. Nous l'avons choisi, car il s'agit d'un standard défini par le W3C pour le développement du Web Sémantique. Ce langage est une extension de RDF (Ressource Description Framework), un modèle représentant les ontologies sous forme de graphe.

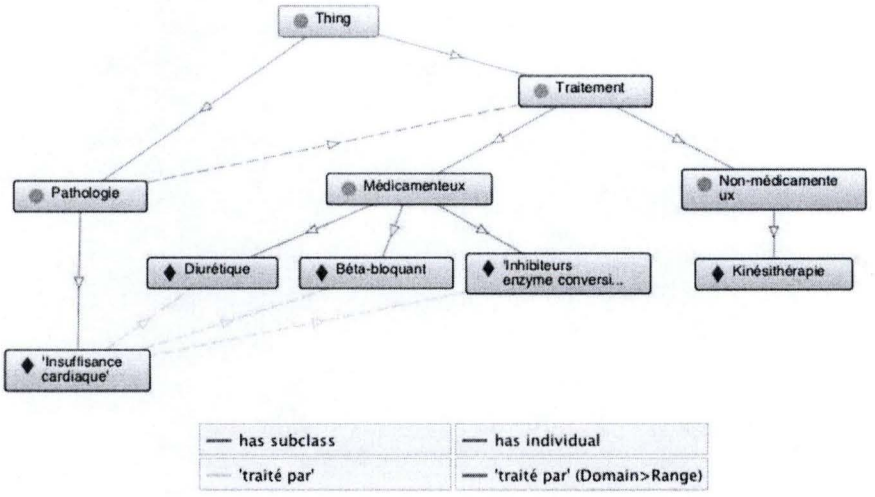


Figure 36 Représentation graphique d'une ontologie RDFS

Un graphe RDF est décrit à l'aide de triplets. Un triplet est une **déclaration** (rdf :Statement) définie en combinant un **sujet** (rdf:subject), un **prédicat** (rdf:predicate) et un **objet** (rdf:object). Par exemple (Diabète, symptôme, Infection). Le sujet identifie la **ressource** que l'on veut décrire. Le **prédicat** est une **propriété** de la **ressource**. L'**objet** est une valeur que l'on attribue à la **propriété**. Le **sujet**, le **prédicat** et l'**objet** sont des **ressources**. Une **ressource** (rdf:Ressource) est caractérisée par un identifiant, il s'agit d'un URI qui permet de l'identifier (rdf:id). Un **littéral** est un type de **ressource** permettant de représenter une chaine de caractère ou un entier.

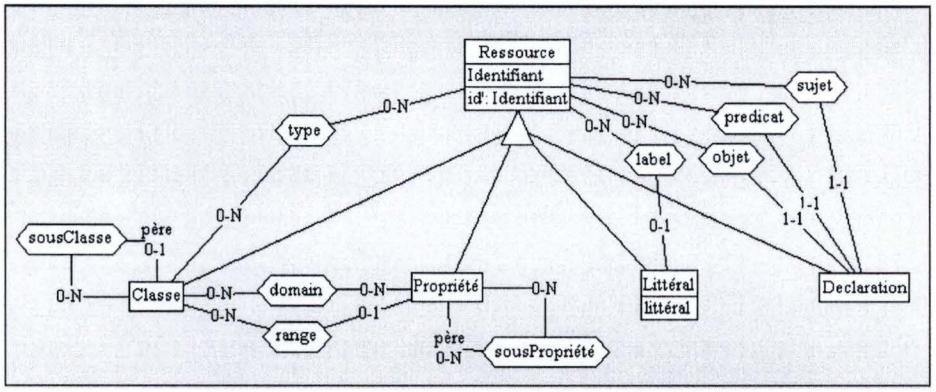


Figure 37 Modèle RDFS

RDFS (W3C, 2004) spécifie un vocabulaire RDF permettant de définir des **classes**, des **propriétés**, des relations d'héritage entre classes et des propriétés. Une **classe** (rdf:class) représente un *concept* dans une ontologie. Les *individus* sont modélisés sous forme de **ressource**. On peut définir qu'une **ressource** est l'instance d'une ou plusieurs classes en utilisant le **prédicat** rdf:type entre une **ressource** et une **classe**. Une **classe** peut être la **sous-classe** (prédicat rdfs:subClassOf) d'une autre **classe**. Une **propriété** (rdf:Property) permet de typer les prédicats. Pour cela, on définit son ou ses **domaines** de définitions (**domain**) (prédicat rdfs:domain) et/ou son ensemble d'arrivée (**range**) (prédicat rdfs:range). Une **propriété** peut être la **sous-propriété** (rdfs:subPropertyOf) d'une autre propriété. Une *relation* entre *concepts* ou *individus* est modélisée par une **déclaration** où le **sujet** est le premier membre de la relation, l'**objet** le second et le **prédicat** désigne la *relation* (de type **Propriété**). Un **label** est un **littéral** décrivant de manière explicite une **ressource**. Une **ressource** peut être dénotée par plusieurs **labels** (prédicat rdf:label).

Transformations

Nous allons voir comment une instance du modèle RDFS représenté à la figure Figure 37 peut être décrite dans un formalisme respectant le modèle de la métabase.

Le langage utilisé pour définir les transformations est détaillé dans la section 2.3.1. Nous reprenons ici les notations utilisées pour décrire les structures du modèle la métabase.

```
INFO_ENTITY(oid, name, type, typeParameters, medium, location) ;
PROPERTY(name, type, ?value) ;
object_attribute((INFO_ENTITY) entity, (PROPERTY) attibute) ;
info-relation(type, (INFO_ENTITY) merber1, (INFO_ENTITY) merber1) ;
composition(role, cardinality, order, (INFO_ENTITY) whole, (INFO_ENTITY) part) ;
```

On ne va pas représenter la syntaxe d'un schéma RDF, mais plutôt la sémantique (voir section « Modélisation selon la sémantique » chapitre 2.3). Ainsi un **schéma RDF** est transformé en entité d'information de type « Ontologie ».

RDF_Schema schema = RDF_Schema(nom, localisation) ;

⇔

INFO_ENTITY schema_ent = INFO_ENTITY(schema.nom, schema.nom, 'Ontologie', _ _ localisation) ;

Une **classe** est enregistrée dans une **entité d'information** de type « Concept ». On lie le **concept** au **schéma** *schema* qui le contient.

Classe concept = Classe(identifiant) ;

⇔

INFO_ENTITY concept_ent = INFO_ENTITY(concept.contient.nom · concept.identidiant, concept.identifiant, 'Concept, _ _ _) ;

info-relation('contient', schema_ent, concept_ent) ;

où *schema_ent* référence une entité d'information représentant *schema.*

Une **ressource** représentant l'instance d'un *concept* est transformée en unité d'information de type « individu ».

Ressource individu = Ressource(identifiant) ;

⇔

INFO_ENTITY individu_ent = INFO_ENTITY(individu.contient.nom · individu.identidiant, individu.identifiant, 'Individu', _ _ _) ;

Une **propriété** est transformée en entité d'information. On lie la **propriété** au **schéma** *schema* qui le contient.

Propriete propriete = Propriete(Identifiant) ;

⇔

INFO_ENTITY proriete_ent = INFO_ENTITY(propriete.contient.nom · propriete.identidiant, propriete.identifiant, 'Propriété', _ _ _) ;

info-relation('contient', schema_ent, propriete_ent) ;

où *schema_ent* référence une entité d'information représentant *schema.*

Chaque **littéral** est transformé en une **unité d'information** ayant pour **valeur** la chaine de caractère représentant le **littéral.** On lie le **littéral** au **schéma** *schema* qui le contient.

Litteral litteral = Litteral(identifiant, litteral) ;

⇔

INFO_UNIT litteral_ent = INFO_UNIT(litteral.contient.nom · litteral.identidiant, litteral.identifiant, 'Litteral', _ _ _ litteral.litteral) ;

info-relation('contient', schema_ent, litteral_ent) ;

où *schema_ent* référence une entité d'information représentant *schema.*

Chaque **déclaration** est transformée différemment en fonction du type de prédicat ;

Declaration instanceDecl = Declaration(sujet, predicat, objet) ;

Si le prédicat est « rdf:type », *sujet* est une **ressource** représentant une instance du concept représenté par l'*objet* de type **classe**. On enregistre cette relation dans une **info relation** de type « type ».

info-relation('type', sujet_ent, objet_ent) ;

où *sujet_ent* référence une entité d'information représentant *sujet* ; objet_ent référence une entité d'information représentant *objet*.

Si le prédicat est « rdfs:subClassOf », il s'agit d'une **déclaration** de subsomption entre une **classe** référencée par *sujet* et une **classe** la subsumant, référencée par *objet*. On représente cette relation dans une **info relation** de type « subsume ».

info-relation('subsume', objet_ent, sujet_ent) ;

où *sujet_ent* référence une entité d'information représentant *sujet* ; objet_ent référence une entité d'information représentant *objet*.

Si le prédicat est une ressource de type « rdfs:Property», la déclaration représente un lien sémantique identifié par *prédicat* entre *sujet* et *objet*. La relation est représentée par une **entité d'information**. Le lien entre la relation et son sujet, son prédicat et son objet sont transformés en **info relation** dont le type est respectivement : « sujet », « prédicat », « objet ».

INFO_ENTITY relation_ent = INFO_ENTITY(predicat.contient.nom · predicat.identidiant · sujet.indetifiant · objet.identifiant, predicat.identifiant, 'relation', _, _, _) ;

info-relation('sujet', relation_ent, sujet_ent) ;

info-relation('prédicat', relation_ent, predicat_ent) ;

info-relation('objet', relation_ent, objet_ent) ;

où *sujet_ent* référence une entité d'information représentant *sujet* ; objet_ent référence une entité d'information représentant l'objet ; predicat_ent référence une entité d'information représentant *predicat*.

Si le prédicat est « rdfs:subPropertyOf», il s'agit d'une **déclaration** de subsomption entre une **propriété** référencée par *sujet* et une **propriété** la subsumant, référencée par *objet*. On représente cette relation dans une **info relation** de type « subsume ».

info-relation('subsume', objet_ent, sujet_ent) ;

où *sujet_ent* référence une entité d'information représentant *sujet* ; objet_ent référence une entité d'information représentant *objet*.

Si le prédicat est « rdfs:domain», *sujet* est une **propriété** dont le domaine de valeur est représenté par *objet* qui est de type **classe**. On enregistre cette déclaration dans une **info relation** de type « domaine ».

info-relation('domaine', objet_ent, sujet_ent) ;

où *sujet_ent* référence une entité d'information représentant *sujet* ; objet_ent référence une entité d'information représentant *objet*.

Si le prédicat est « rdfs:range», *sujet* est une **propriété** dont l'intervalle de valeur est représenté par *objet* qui est de type **classe**. On enregistre cette déclaration dans une **info relation** de type « intervalle ».

info-relation('intervalle', objet_ent, sujet_ent) ;

où *sujet_ent* référence une entité d'information représentant *sujet* ; objet_ent référence une entité d'information représentant *objet*.

Si le prédicat est « rdfs:label», *sujet* est une **ressource** et *objet* un littéral ayant pour valeur une chaine de caractère. On enregistre ce lien dans une **info relation** de type « domaine ».

info-relation('label', objet_ent, sujet_ent) ;

où *sujet_ent* référence une entité d'information représentant *sujet* ; objet_ent référence une entité d'information représentant *objet*.

Le schéma ci-dessous représente l'ontologie de la Figure 36 exprimée dans le modèle de la métabase. Les concepts sont représentés en jaune.

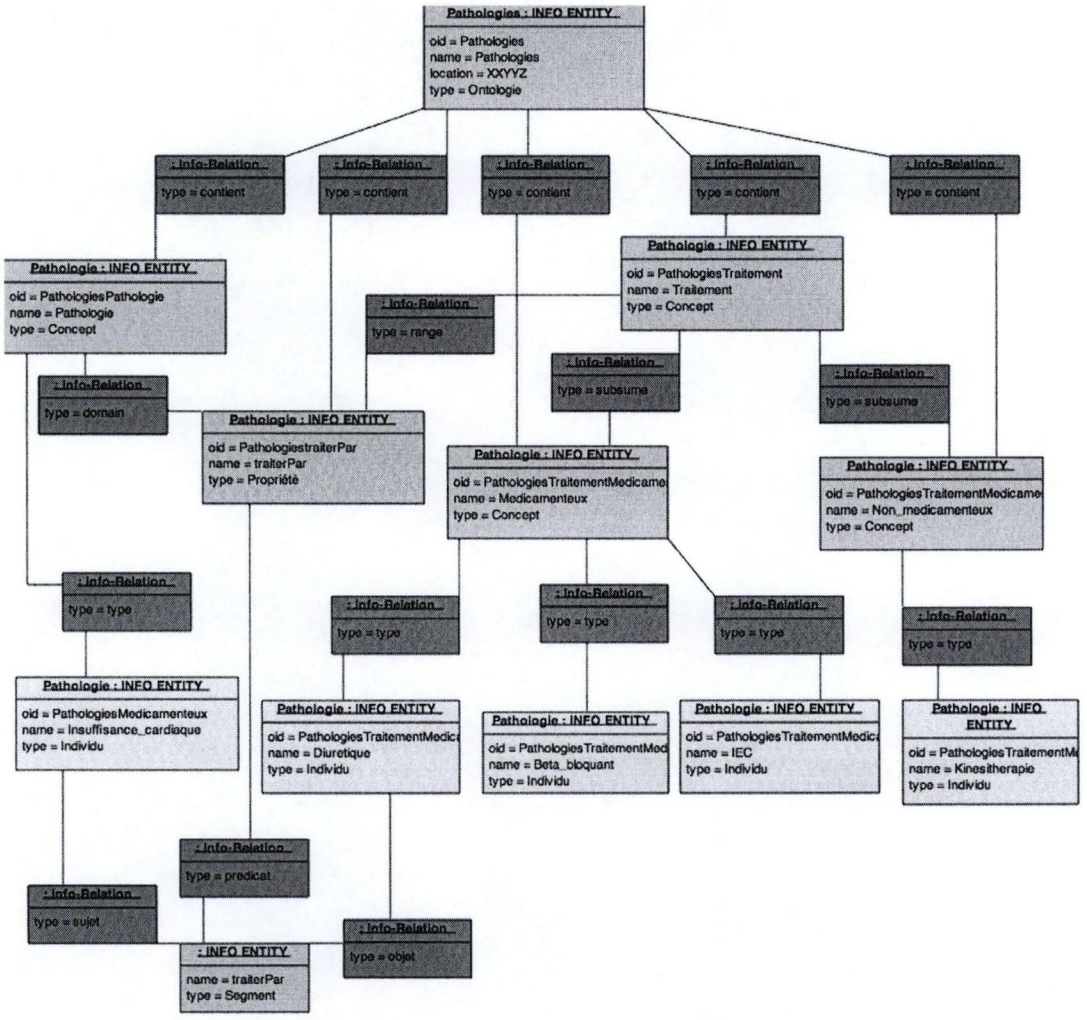


Figure 38 Ontologie exprimée en utilisant le vocabulaire de la métabase

2.8. Représentation des correspondances entre les données

Il est nécessaire de pouvoir représenter les correspondances qui existent entre les différents formalismes abordés lors des précédents chapitres. Il y a correspondances entre deux composants, appartenant à des descriptions différentes, lorsque ceux-ci décrivent la même entité du monde réel. Nous allons voir comment et pourquoi il est nécessaire de garder une trace de ces correspondances dans la base de métadonnées. Nous distinguerons deux types de liens : verticaux entre descriptions dérivant l'une de l'autre et horizontaux entre composants décrivant les mêmes entités du monde réel.

2.8.1. Verticales conceptuel / logique dans les BD et Schéma XML

Les correspondances verticales décrivent la même portion du domaine d'activité, mais avec des niveaux de précision différents. Une telle correspondance peut exister entre un schéma conceptuel et un schéma logique de base de données, un schéma conceptuel et un XML schéma, un XML schéma et un document XML s'y conformant, ... Ces correspondances existent lorsque des modèles sont issus du même processus. Une partie des modèles est donc issue de la transformation d'autres modèles. On distingue deux types de correspondance verticale. Un premier type, décrit aux cours de précédents chapitres, permet de lier les descriptions ou modèles aux instances représentés par des index. Les secondes, auxquelles nous nous intéressons ici, concernent les relations entre descriptions.

Dans le chapitre consacré à la modélisation des bases de données (chapitre 2.6), nous avons vu que le développement d'une base de données pouvait être vu comme un processus prenant la description des besoins utilisateurs en entrée qu'il traduit en code DDL permettant de générer la base de données. Ce processus était lui-même composé de sous-processus transformant un schéma en entrée en un autre schéma. Au fil des transformations, on ajoute des détails spécifiques à l'implémentation. On s'éloigne d'une description compréhensible par l'homme pour arriver à une description implémentable sur un système particulier. Ces processus utilisent des règles pour transformer les composants du schéma fournis en entrée. Il existe donc un lien entre chaque élément composant le schéma source et le résultat après application de la règle. Ce sont ces liens que l'on souhaite représenter dans la métabase.

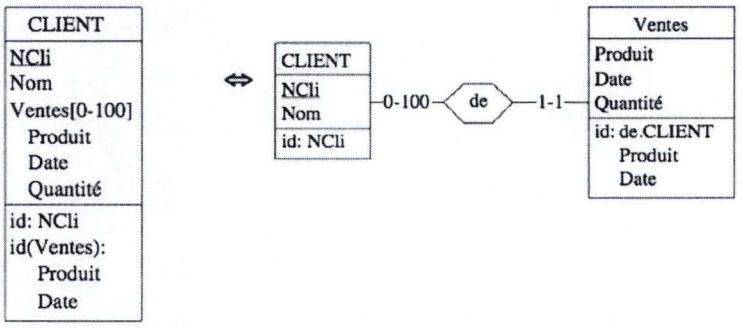


Figure 39 Transformation d'un attribut composé multivalué en TE (Hainaut, 2003)

La figure ci-dessus montre la transformation d'un attribut composé multivalué « Ventes » en un TE. Dans ce cas, il existe un lien entre le TE « Client » et les TE « Client » et « Ventes » résultants de la transformation.

Plus généralement, les processus de développement de sources de données partent d'une description de haut niveau qui formalise les besoins utilisateurs pour arriver à une description de bas niveau permettant d'implémenter ses besoins sur une

plateforme spécifique. Les schémas de hauts niveaux sont utiles à notre démarche, car ils font abstraction de tous détails spécifiques à l'utilisation de technologies, en vogue à un moment donné. Ils resteront valables en cas de changement de technologies ou si plusieurs implémentations d'un même schéma de haut niveau existent. Les schémas de bas niveaux sont indispensables, car grâce aux détails d'implémentation, ils permettront d'exploiter les sources de données qu'ils décrivent.

Il existe plusieurs façons de mémoriser ces correspondances dans la métabase, initialement prévue pour modéliser l'évolution d'un workflow, nous avons retenus trois approches décrites dans (Hainaut, Brogneaux, & Cleve, 2010) : l'approche transformationnelle, l'approche par correspondance et l'approche par estampillage. Ces procédés peuvent être appliqués à toutes entités héritant de WF Object. Ils sont donc utilisables avec les entités de type « Info Entity » et « Info Unit » utilisées jusqu'ici.

Considérons deux descriptions D1 et D2 où D2 est issus de la transformation de D1. Afin de garder les notations utilisées dans (Hainaut, Brogneaux, & Cleve, 2010), nous appellerons W2 une composante de D2 issue de la transformation de la composante W1 dans D1. Un processus de transformation pouvant produire plusieurs éléments à partir d'un seul ou à l'inverse réduire le nombre d'éléments du modèle de départ, W1 et W2 sont des composants contenant un ou plusieurs objets.

L'approche transformationnelle permet d'enregistrer la liste des opérations ayant permis de produire W2 à partir de W1. Une transformation a eu lieu à une certaine date, en utilisant un opérateur de transformation et une liste de paramètres. Elle concerne un ensemble de composants (« transfo-objects ») fournis en entrée ou en sortie. Bien que très complète, le nombre d'informations à fournir rend l'application de cette méthode contraignante.

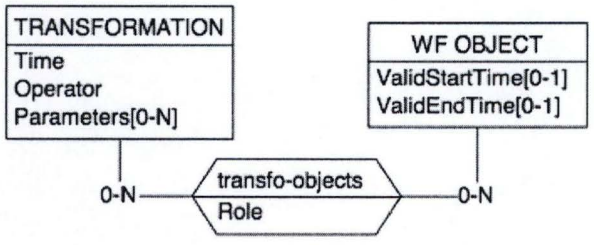


Tableau 1 Modélisation des correspondances selon l'approche transformationnelle (Hainaut, Brogneaux, & Cleve, 2010)

L'approche par correspondance consiste à n'enregistrer que le résultat du processus de transformation. Il permet ainsi de définir une relation « derive » entre des objets W1 source et les résultats W2.

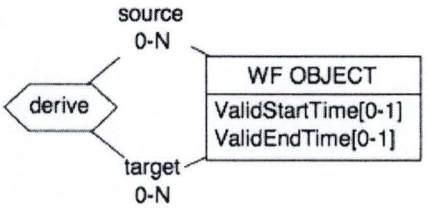


Tableau 2 Modélisation des correspondances selon l'approche par correspondance (Hainaut, Brogneaux, & Cleve, 2010)

L'approche par estampillage consiste à faire hériter les composantes de W2 de l'estampille des composantes de W1. Une estampille peut prendre la forme d'un numéro, unique à chaque composant du modèle de départ. Si W1 est composé d'un seul élément, le ou les éléments de W2 porteront tous ce même numéro. À l'inverse si la transformation de W1 ne produit qu'un élément dans W2, ce dernier héritera des estampilles de tous les éléments présents dans W1. On peut stocker ses estampilles sous forme d'une liste de « PROPERTY » qui constitue les « attributs » de l'entité « Object ». Cette approche a l'inconvénient d'être moins expressive. On perd les avantages de l'intégrité référentielle. On peut donc se retrouver avec des informations incohérentes où une estampille pour W2 n'a aucune correspondance dans W1.

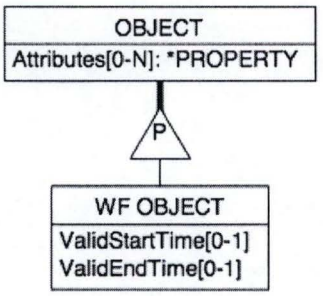


Tableau 3 Modélisation des correspondances par estampillage (Hainaut, Brogneaux, & Cleve, 2010)

Le schéma ci-dessous présente un exemple de correspondance entre une partie du schéma conceptuel et logique, représenté respectivement à la Figure 32 et la Figure 35.

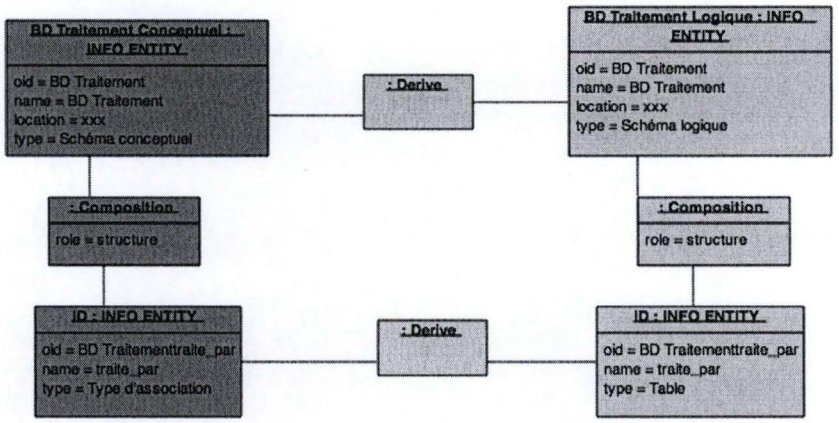


Figure 40 Représentation des correspondances verticales entre un schéma conceptuel et logique

On a choisi d'exprimer les correspondances à l'aide du TA « derive » pour exprimer le fait que le schéma logique est issu de la transformation du schéma conceptuel.

2.8.2. Horizontales entre descriptions conceptuelles

Les correspondances horizontales existent lorsque des composants appartenant à des modèles différents décrivent une même entité du monde réel. Ces modèles peuvent décrire les mêmes entités du modèle d'activité en utilisant des formalismes différents. De telles correspondances peuvent exister entre une ontologie et un schéma conceptuel, un XML Schema, les termes d'un document « plat », ...

Lier une ontologie aux descriptions des sources de données permet d'ajouter une dimension sémantique aux données contenues dans la métabase. En effet, les différentes descriptions ayant été conçues dans des contextes et avec des objectifs différents, des concepts identiques, dans le monde réel, seront modélisés différemment. Une ontologie permet d'uniformiser la spécification d'un domaine d'activité, commun aux différents documents. Le domaine devra donc englober celui des données présentes dans la métabase. Pour décrire le sens d'une granule, on définit des correspondances entre les ressources de l'ontologie, c'est-à-dire ses concepts et ses individus, et les composants des sources de données.

On distingue deux types de relations entre ontologies et sources de données : « référence » et « représente ». Le contenu d'une source de donnée fait **référence** à un concept de l'ontologie alors qu'un élément de sa structure **représente** un concept de l'ontologie. Ainsi, un concept de l'ontologie sera lié aux index des documents « plats, des documents semi-structurés et des données structurées par une **info-relation** de type « référence ».

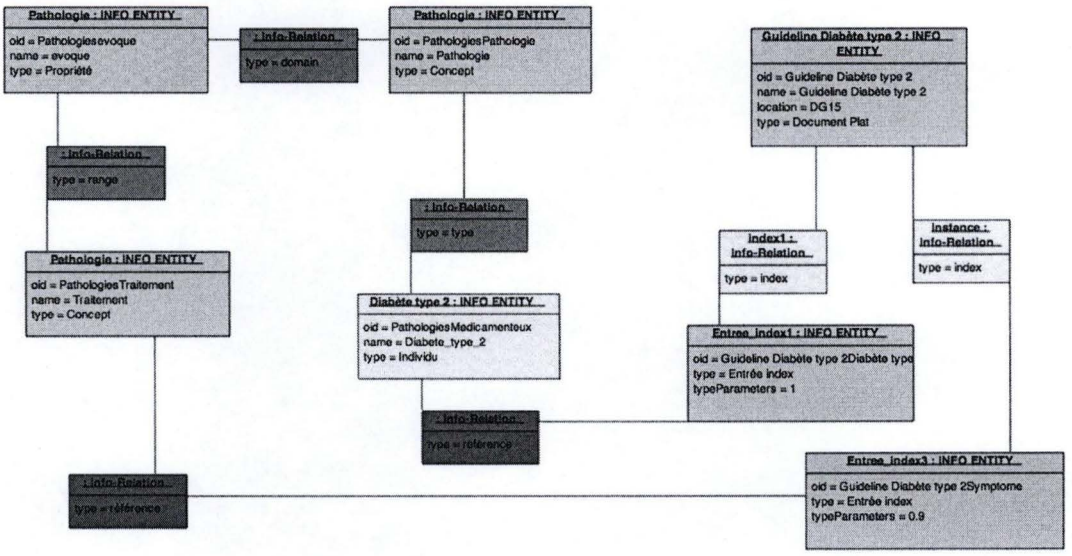


Figure 41 Correspondances verticales entre un document (droite) et les concepts de l'ontologie auquel il fait référence.

Un concept de l'ontologie sera lié à une déclaration d'élément XML ou une table de base de donnée par une **info-relation** de type « représente ».

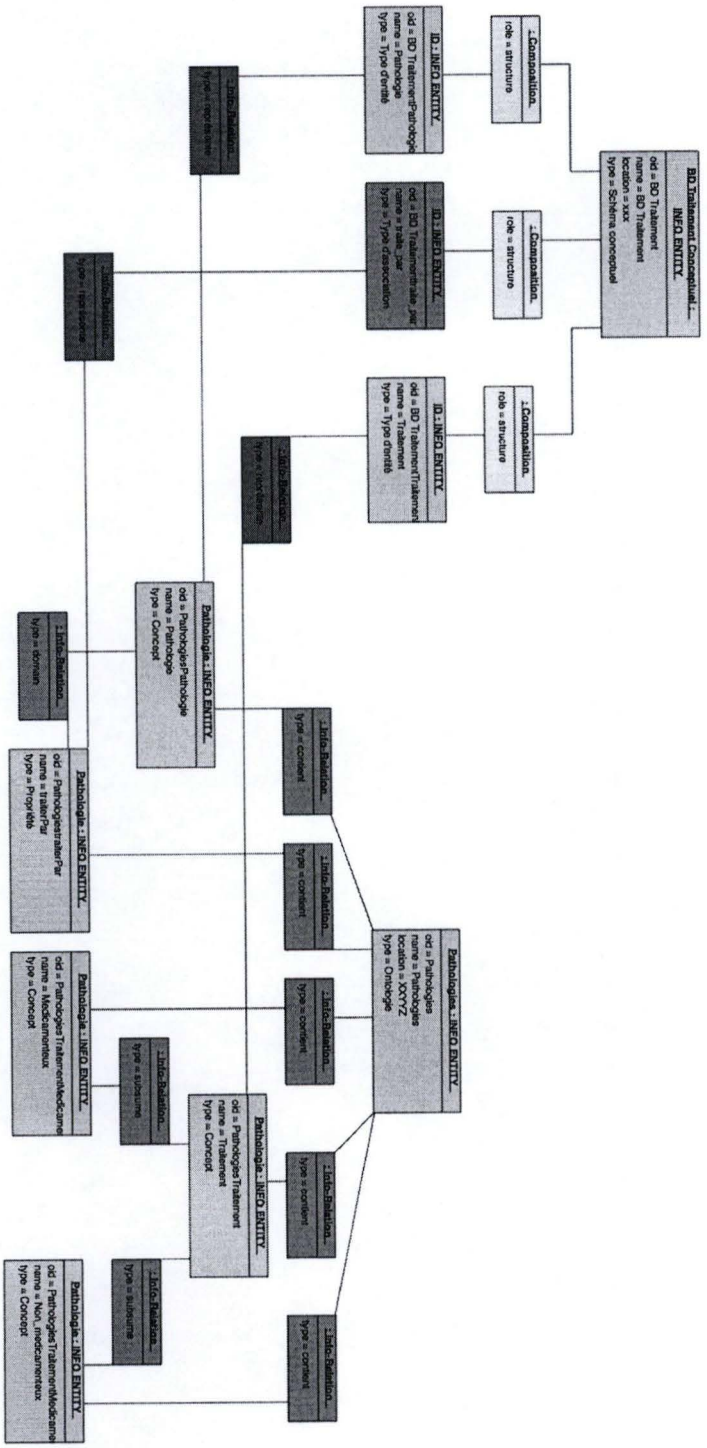


Figure 42 Correspondances entre un schéma conceptuel et les concepts de l'ontologie qu'ils représentent

Un document texte ne fait que parler de différents sujets dénotés par les concepts de l'ontologie. La table « Pathologie » d'une base de données représente la même chose que le concept, du même nom, dans l'ontologie ; tous deux représentent les mêmes objets du monde réel ; en plus de faire référence à ce concept, la table « Pathologie » est susceptible de contenir des instances de ce concept.

Comme on le voit dans le schéma ci-dessus, pour décrire la structure d'une base de données, on liera les l'ontologie au schéma conceptuel. En effet, comme on l'a vu dans le chapitre 1.3.4, consacré aux similitudes entre ces deux formalismes, il sera plus facile d'automatiser le processus de mise en relation, du fait de leurs ressemblances. Concernant le schéma logique, on pourrait retrouver les correspondances avec l'ontologie via le schéma conceptuel dont il dérive ; mais pour faciliter l'exploitation des descriptions de base de données, sujet du prochain chapitre, on introduira de la redondance. Ainsi les tables et attributs du schéma logique seront directement mis en correspondance avec l'ontologie.

2.9. Conclusion

Dans ce chapitre, nous avons vu comment représenter les descriptions de documents « plats », de documents semi-structurés et de données structurées afin de les stocker dans une base de données commune, surnommée métabase. Une description décrivant à la fois le contenu du document, sous forme d'un index, et sa structure, décrite dans un modèle.

Nous avons commencé par décrire les constructions génériques du schéma développé dans le cadre du projet GISELE et qui ont constitué le schéma de notre métabase.

Après avoir décrit les différentes approches permettant de stocker un modèle dans la métabase selon sa syntaxe ou selon sa sémantique ; nous avons défini des règles permettant de stocker une description de documents « plats », XML et de base de données dans la métabase. Tous ces éléments peuvent être utilisés pour construire le schéma d'une nouvelle base de données, contenant des constructions spécifiques aux descriptions que l'on veut y stocker.

Finalement, nous avons montré comment modéliser les correspondances entre différents types de descriptions dans la métabase. Les correspondances verticales permettent de lier une description à celle dont elle dérive par un processus de transformation. Les correspondances horizontales sont particulièrement intéressantes, car elles permettent de donner du sens aux descriptions en les liant avec des formalismes de représentation de connaissances comme les ontologies.

Dans le chapitre suivant, nous verrons comment exploiter ces descriptions pour retrouver de l'information dans les documents qui y sont représentés.

Chapitre 3

Interrogation de données hétérogènes

3.1. Introduction

Lors du précédent chapitre, nous avons discuté de la manière de représenter les descriptions de documents « plats », documents semi-structurés et données structurées dans une métabase. Nous allons maintenant discuter de l'exploitation de ces descriptions pour interroger les sources de données qu'elles représentent.

Le but à long terme est de disposer d'un système de Recherche d'Informations, pour la plateforme e-Health, permettant de retrouver des informations dans des sources de données hétérogènes ; et ce, de manière transparente pour l'utilisateur. Sur base des critères de recherches défini par l'utilisateur ; on souhaite lui présenter la liste des documents « plats » répondant à ces critères ; suivie de la liste des granules pertinentes dans les documents semi-structurés ; et finalement la liste des résultats pertinents provenant de l'interrogation des bases de données.

Dans ce chapitre, on se contentera de localiser les granules pertinentes dans les descriptions, en tentant d'aller le plus loin possible dans la réalisation de l'objectif du système.

Pour cela, on se servira d'une ontologie du domaine pour identifier les besoins en informations de l'utilisateur, en terme de concepts. On suppose que la sémantique des descriptions est encodée dans la métabase sous forme de liens entre leurs granules et les concepts de l'ontologie ; comme nous l'avons vu dans le chapitre 2.8.2. Les relations sémantiques entre les concepts nous permettrons d'enrichir la requête initiale de l'utilisateur.

Nous commencerons par discuter des langages de requêtes afin de permettre à l'utilisateur d'exprimer son besoin en information le plus efficacement possible. Nous verrons ensuite comment identifier les concepts de l'ontologie, pertinents par rapport à la requête. On verra ensuite comment les relations entre ces concepts peuvent être utilisées pour enrichir la requête. Une fois tous les concepts pertinents identifiés, nous verrons comment localiser les documents y faisant référence. Pour finir, nous verrons comment interroger les sources de données contenant les documents « plats », documents semi-structurés et données structurées.

3.2. Langage de requêtes

Plusieurs approches différentes permettent à l'utilisateur d'exprimer ses besoins en informations. La première est d'utiliser un langage issu du monde des bases de données (requête orientée structure). La deuxième est d'utiliser un langage libre (requête orientée contenu). Une troisième beaucoup plus simple à mettre en place, et qui semble prometteuse est de permettre à l'utilisateur de sélectionner directement les concepts de l'ontologie.

Langage orienté structure

Les langages orientés structure se servent du fait que la structuration des données ou des documents est connue à l'avance pour exprimer la requête. Cela permet de spécifier des conditions précises sur les valeurs que doivent prendre certaines granules (nœuds en XML et colonnes en BD) pour être pertinentes. En SQL, par exemple, on spécifie des conditions du type, *colonne = 'valeur'*. L'autre spécificité de ce type de langage est que l'on exprime le niveau de granularité de l'information à retourner dans la requête. Malgré que ce type de langage permette d'exprimer très précisément la forme du résultat, il n'est pas adapté à notre cas où l'utilisateur n'a pas de connaissance poussée en informatique. Tout d'abord parce que ces langages sont très formels et nécessitent une période d'apprentissage avant de pouvoir formuler la moindre requête. Ensuite parce que pour pouvoir exprimer toutes ces contraintes, il faut que l'utilisateur sache comment sont structurées les données, ce qui complexifie encore la tâche de l'utilisateur. Il y a aussi le problème des documents non structurés pour lesquels un langage orienté structure n'a aucun intérêt. Finalement, les langages issus du monde des BD se basent sur un modèle booléen, ce qui implique qu'un élément doit satisfaire entièrement à la requête pour être sélectionné. Les modèles issus de la RI utilisent des mesures de la pertinence pour constituer leurs résultats. Cela permet de classer les résultats selon cette mesure et surtout de renvoyer des données qui ne remplissent pas tous les critères, mais restent pertinentes.

Langage orienté contenu

Un langage orienté contenu permet d'exprimer des requêtes composées d'une liste de mots-clefs. Au vu des inconvénients des langages orientés structure, un tel langage est plus pertinent pour notre système qui s'adresse à des utilisateurs moyens et doit permettre d'interroger à la fois des données structurées, semi-structurées et non structurées. L'utilisateur formule sa requête en spécifiant une liste de mots clefs qu'ils souhaitent retrouver dans les documents que le système va lui renvoyer.

Sélection des concepts dans l'ontologie

Une autre possibilité est de permettre à l'utilisateur d'exprimer directement sa requête en terme de concepts de l'ontologie. Sa requête ne sera plus composée de mots-clefs, mais de concepts. Pour cela, il faut lui proposer un moyen de composer sa liste de concept.

La première solution, vue au chapitre 1.1.3, est d'afficher la hiérarchie de concepts pour qu'il puisse sélectionner ceux qui l'intéressent ; ces concepts seront alors ajoutés à la requête au fur et à mesure de leurs sélections par l'utilisateur. Cela permet d'aider l'utilisateur qui n'a pas d'idée précise du vocabulaire à employer pour exprimer son besoin en information. La hiérarchie de concept peut facilement être représentée graphiquement sur base des concepts de l'ontologie et des relations de subsomptions qui les lient. Le problème est que la définition d'une requête peut devenir fastidieux.

Une deuxième solution consiste à sélectionner les concepts grâce à un mécanisme d'autocomplétion. Au fur et à mesure de la saisie de caractères par l'utilisateur, on affiche une liste de concepts correspondants ; soit un seul concept correspond aux termes rentrés jusqu'ici, on ajoute ce concept à la requête ; soit plusieurs concepts correspondent auquel cas : il peut sélectionner l'un d'entre eux ou continuer à entrer des caractères.

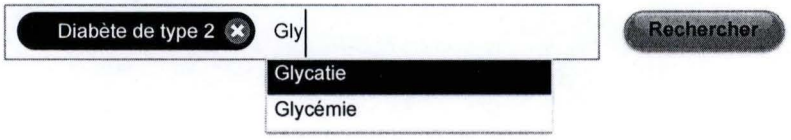


Figure 43 Interface permettant de sélectionner des concepts via un mécanisme d'autocomplétion

Le fait de sélectionner les concepts directement sans passer par une requête composée de mots-clefs permet d'éviter tous les problèmes d'ambiguïté décrits dans la section suivante. Un tel système est donc beaucoup plus facile à implémenter.

3.3. Identification des concepts

Cette étape vise à identifier les concepts dans l'ontologie auxquels l'utilisateur a voulu faire référence, avec les termes composant sa requête. Plus formellement, cette étape consiste à faire correspondre les termes de la requête avec les concepts de l'ontologie du domaine. Elle ne concerne donc que les requêtes exprimées sous forme d'une liste de mots-clefs. Au terme de cette étape, nous aurons réalisé l'appariement entre la requête et les documents, étant donné que ces derniers ont eux aussi été mis en correspondance avec l'ontologie lors de leurs indexations. Nous allons d'abord considérer le cas où la requête est composée d'un seul syntagme c'est-à-dire le cas où

le ou les termes ne dénotent donc qu'un seul concept. Les requêtes 'diabète de type 2' et 'symptôme' ne sont composés que d'un seul syntagme. Nous parlerons ensuite des techniques permettant de repérer les syntagmes dans un ensemble de termes. La requête 'symptôme diabète de type 2' est composée de deux syntagmes si l'on considère que l'on recherche les symptômes du diabète de type 2.

Requête simple

Une requête simple est une requête composée d'un seul syntagme. Un syntagme dénote un concept. Cela ne veut pas dire qu'elle n'est composée que d'un seul terme, mais que les termes forment une même unité de sens. La requête 'diabète de type 2' est composée de plusieurs termes, mais ne fait référence qu'a un concept dans l'ontologie, représentant la maladie auquel cette connotation fait référence.

Nous allons utiliser les labels associés aux concepts de l'ontologie pour trouver le concept correspondant aux termes de la requête. Pour rappel, un label est une description en langage naturel d'un concept de l'ontologie. On considère donc que toutes les connotations des concepts de l'ontologie sont décrites par ces labels. Cela permet aussi de prendre en compte plusieurs langues en étiquetant un concept avec ses connotations dans toutes ces langues. On peut ainsi rechercher le concept en comparant les termes de la requête aux labels. En reprenant l'exemple précédent, on cherchera le label 'diabète de type 2' pour trouver le concept représentant la maladie auquel il est lié. Lorsque la requête est composée de plusieurs termes, il faut retrouver ces termes dans le même ordre pour qu'il y ait correspondance. Ainsi les requêtes 'radiographie thorax' et 'thorax radiographie' ne correspondront pas aux mêmes concepts. Au terme de ce processus, trois cas sont possibles :

- aucun concept ne correspond,
- un seul concept correspond,
- plusieurs concepts correspondent.

Le premier cas peut être dû à un syntagme mal orthographié, une connotation qui n'a pas été prise en compte (c.-à-d. un label manquant) ou un concept absent de l'ontologie. Le second cas, où un seul concept de l'ontologie correspond, représente le cas idéal : on a trouvé le concept auquel l'utilisateur souhaitait faire référence. Dans le dernier cas, le syntagme désigne plusieurs concepts de l'ontologie. La probabilité que ce cas se présente reste très faible lorsqu'on limite le SRI à un seul domaine d'activité, plus encore lorsqu'il s'agit de médecine. En effet, pour être efficace, le vocabulaire lié à un domaine d'activité doit éviter les risques d'ambiguïté. Il serait très risqué pour le patient que les termes utilisés dans une prescription puissent être interprétés de différentes manières. Néanmoins, des cas de polysémie peuvent être rencontrés avec les abréviations de termes médicaux. Ainsi l'abréviation PAC peut désigner une voie centrale (porth-a-cath) ou un pontage aorto-coronarien.

Lorsqu'un cas de polysémie survient, plusieurs techniques permettent de désambiguïser les labels. Dans (Guha, McCool, & Miller, 2003), on trouve trois critères pour déterminer le concept visé par l'utilisateur. Le premier consiste à prendre en compte la fréquence d'apparition du concept dans le corpus. Ils expliquent que pour la requête « Paris », la ville de Paris en France sera un meilleur choix que la ville Paris au Texas ou le groupe de musique Paris qui apparaissent moins souvent. Le deuxième critère consiste à prendre en compte le profil de l'utilisateur. On peut l'appliquer à notre cas en prenant en se servant de la spécialité du médecin utilisant le système. Le dernier critère est de se servir du contexte de recherche. Les informations de contexte peuvent être tirées des précédentes recherches, mais aussi en fonction de la distance sémantique séparant l'ensemble des concepts visé par la requête. Nous allons maintenant nous intéresser au cas des requêtes complexe où nous verrons comment fonctionnent les mesures de désambiguïsations basées sur la distance sémantique entre concepts.

Requête complexe

Nous nous basons sur la démarche décrite dans (Baziz, Boughanem, Aussenac-Gilles, & Chrisment, 2005) pour la détection des syntagmes. Le principe est de favoriser les labels les plus longs, car dans les faits, au plus un groupe nominal est long au plus la probabilité qu'il ait plusieurs sens est faible. Pour cela, on projette les termes de la requête sur l'ontologie en faisant varier la taille d'une fenêtre sur les termes de la requête (au début, la fenêtre porte sur tous les termes de la requête). On cherche le syntagme le plus long formé par des termes adjacents et appartenant à au moins une entrée dans l'ontologie.

Une fois que les groupes nominaux ont été identifiés, il est possible de les utiliser comme informations de contexte pour désambiguïser ceux qui possèdent plusieurs sens. L'idée est de favoriser les combinaisons de concept sémantiquement proche comme expliqué dans (Navigli & Velardi, 2003). Pour chaque combinaison possible, appelée configuration, on calcule une mesure de similarité entre chaque pair de concept. Cela permet d'associer un score à chaque concept. On choisit ensuite la configuration qui maximise ce score. Des mesures de similarités sont décrites au chapitre 1.2.

Même si les techniques de désambiguïsation des labels sont efficaces, elles ne doivent pas remplacer l'utilisateur qui est le seul à connaître le sens de sa requête. Lorsque le SRI se trouve devant un tel cas, l'utilisateur doit en être informé. Une bonne interface de recherche doit :

- indiquer à quels concepts ont été liés les termes
- avertir l'utilisateur en cas de polysémie
- en cas de polysémie, lui permettre de choisir parmi tous les concepts candidats

3.4. Enrichissement de la requête

L'affinement, ou enrichissement, de la requête consiste à affiner la requête de l'utilisateur à l'aide d'une ontologie du domaine. Le but de cette étape est de suggérer des concepts auxquels l'utilisateur n'aurait pas pensé, mais qui sont susceptibles d'êtres pertinents par rapport à son besoin en information. Lors de l'étape précédente, nous avons identifié les concepts de l'ontologie sélectionnés par l'utilisateur. Nous allons pouvoir nous servir des relations qu'entretiennent ces concepts avec d'autres, dans l'ontologie, pour enrichir la requête. Pour cela, nous allons utiliser trois types de relations :

- Relations de spécialisation vont nous permettre d'enrichir la requête de concepts plus spécifiques. Pour un concept A, on cherche des concepts héritant de A.
- Relations de généralisation vont nous permettre d'enrichir la requête en ajoutant des concepts plus généraux. Pour un concept A, on cherche des concepts dont A hérite.
- Relations sémantiques vont nous permettre d'enrichir la requête en exploitant n'importe quel type de relations que le concept partage avec les autres.

Bien qu'il soit possible d'automatiser le processus d'enrichissement de la requête, nous allons utiliser une approche manuelle. En effet, exclure l'utilisateur du processus peut le perturber, car il ne sait pas comment le système a enrichi sa requête. On voit donc plutôt l'enrichissement comme un processus incrémental que l'utilisateur peut diriger en fonction des résultats obtenus ; il doit pouvoir affiner sa requête à tout moment ; il doit pouvoir supprimer les concepts issus de l'enrichissement pour revenir en arrière. Afin d'affiner sa requête, l'interface de recherche devra donc proposer une liste des enrichissements applicables aux différents termes composant de la requête. Dans le but de rendre le processus d'enrichissement transparent à l'utilisateur, il faudra retoucher la requête initiale afin que les nouveaux concepts y soient inclus. Concrètement, on ajoutera un des labels associés à chaque concept rajouté à la requête. Il pourra ainsi supprimer n'importe quel terme et ainsi supprimer des concepts issus d'un enrichissement pour revenir en arrière.

L'enrichissement de la requête peut avoir un impact négatif sur le calcul de la pertinence si l'on applique une pondération uniforme entre les concepts d'origines et les concepts rajoutés. Prenons le cas d'une recherche portant sur les traitements du diabète ; on suppose que le calcul de la pertinence d'une requête par rapport à un document est défini comme le nombre de concepts composant la requête trouvé dans le document sur le nombre total de concepts de la requête.

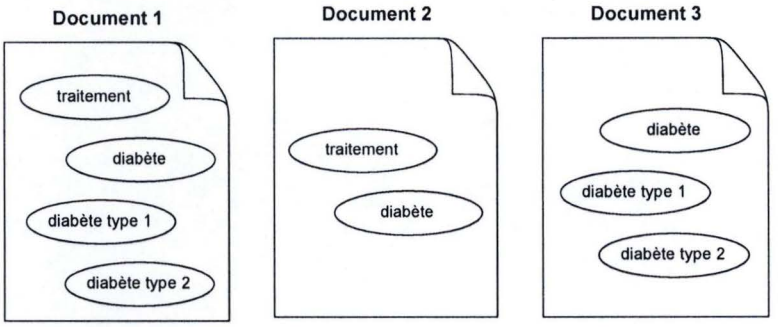


Figure 44 Documents associés à la représentation des concepts qu'ils référencent

Les documents 1 et 2 associés au concept 'diabète' et au concept 'traitement' sont donc à 100% pertinents pour la requête : 'diabète traitement'. Le document 3 ne parlant pas de traitement, mais seulement de diabète n'est pertinent qu'à 50%. Si l'on étant la requête par spécialisation, ajoutant ainsi les instances 'diabète type 1' et 'diabète type 2', la requête devient 'diabète traitement diabète type 1 diabète type 2'. Le document 1 reste à 100% pertinent seulement le document 3 à maintenant une pertinence de 75% et le document 2 qui passe à 50%. Par l'enrichissement, on a diminué le classement d'un document intéressant pour l'utilisateur (selon sa requête initiale) au profit d'un document qui l'est moins. Pour résoudre ce problème, les concepts issus de l'enrichissement et le concept dont ils dérivent auront le même poids que ce dernier avait dans la requête initiale. Dans l'exemple, les concepts 'diabète', 'diabète type 1' et 'diabète type 2' auront ensemble un poids de 50% et 'traitement' gardera son poids de 50%. '(diabète diabète type 1 diabète type 2) traitement'.

Concrètement, si l'on utilise le modèle vectoriel pour le calcul de la pertinence, on assigne un poids a chacun des termes dans la requête ; normalement ce poids est la même pour chaque terme ; Ici, lorsque l'on ajoute un ensemble de concepts CA sur base d'un concept cb, on calcul leurs poids dans la requête comme suit :

$$\forall c \in CT: w_{q,c} = poids_{corr}$$

où CT = CA ∪ {cb} ; $w_{q,c}$ est le poids du concept c dans la requête q; et poids_corr = $\frac{cb}{|CT|}$

Dans la suite, nous allons voir comment exploiter les relations sémantiques entre concepts de l'ontologie pour réaliser l'enrichissement.

3.4.1. Spécialisation

L'enrichissement par spécialisation consiste à enrichir la requête d'un ensemble de concepts plus spécifiques. Pour un concept, on va exploiter les relations de subsomption pour trouver un ensemble de concepts héritant des caractéristiques de

celui-ci. Les relations d'héritage entre concepts forment une structure arborescente où le concept initial est la racine et les feuilles constituent les instances de ces concepts, les individus.

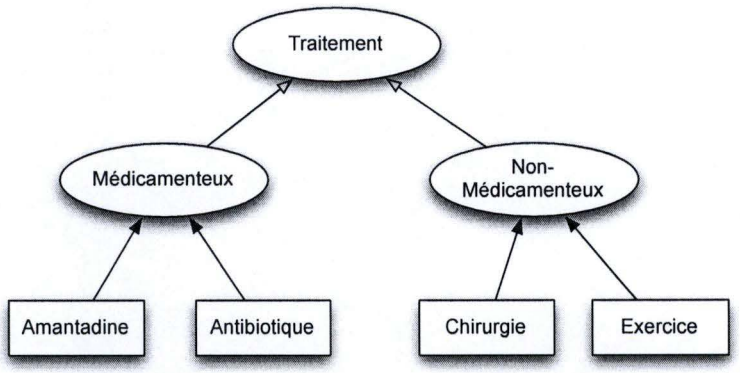


Figure 45 Extrait d'ontologie – enrichissement par spécialisation

Alors que la requête initiale permettait à l'utilisateur de trouver des documents traitant d'un sujet en toute généralité ; L'enrichissement par spécialisation lui permettra de trouver des documents qui se classent dans la même catégorie, mais aborde une partie plus spécifique de ce sujet original. En prenant l'exemple d'ontologie ci-dessus, il est possible d'enrichir le concept « Traitement », des concepts « Médicamenteux » et « Non-médicamenteux » ainsi que toutes leurs instances.

3.4.2. Généralisation

L'enrichissement par généralisation permet d'enrichir la requête de concepts plus généraux. Pour un concept donné, on va retrouver via les relations de subsomption, les concepts qui le subsument. On ne remontera d'un niveau à la fois, en laissant le choix à l'utilisateur d'aller plus loin.

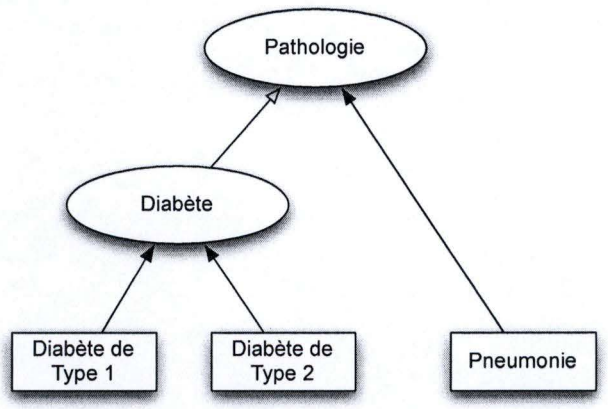


Figure 46 Extrait d'ontologie – enrichissement par généralisation

Par rapport à la requête initiale, qui permettait à l'utilisateur de trouver des documents traitant de ce sujet de manière très spécifiques ; l'enrichissement par généralisation, lui permettra de trouver des documents traitant de la même famille sujet, mais de manière plus générale. En prenant l'exemple d'ontologie ci-dessus, le concept « Diabète de Type 1 » est généralisé par les concepts « Diabète » et « Pathologie ».

3.4.3. Relations sémantiques

Les relations sémantiques peuvent aussi être utilisées pour étendre la requête. Dans ce cas, c'est à l'utilisateur de choisir la relation sémantique via laquelle il souhaite enrichir sa requête pour trouver de nouveaux concepts.

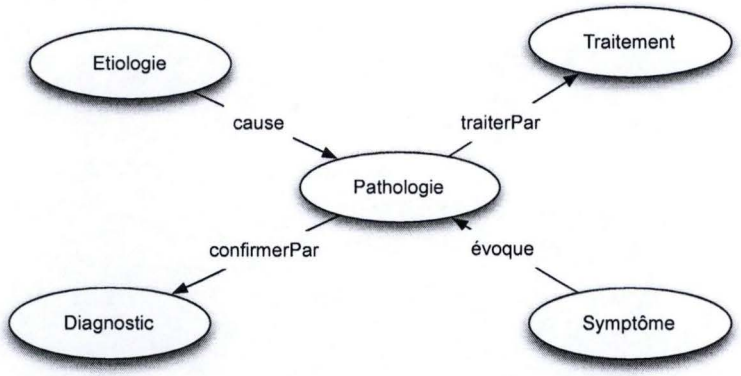


Figure 47 Extrait d'ontologie – enrichissement via les relations sémantiques

En considérant l'ontologie ci-dessus, à partir du concept « Pathologie », on peut par exemple utiliser la relation « confirmerPar » pour enrichir la requête du concept « Diagnostic ».

3.5. Localisation des sources de données pertinentes

Maintenant que nous pouvons identifier à partir de la requête, les concepts intéressant l'utilisateur, nous sommes en mesure de retrouver les documents pertinents. En effet, l'appariement requête/document ayant été réalisé via l'ontologie du domaine, il est maintenant possible d'exploiter ces correspondances pour identifier les documents pertinents.

À partir des concepts sélectionnés via la requête, il faut identifier les sources de données traitant d'au moins un de ces concepts ; mais aussi pour chaque source de donnée, les composants concernés. On distinguera deux cas : soit une référence au concept est faite dans une instance d'une granule de document, soit le concept est représenté par un composant de document.

Lorsque le concept est référencé dans une instance de document ou d'une de ces granules, la correspondance est directe. Si un document texte parle de la pneumonie, le concept représentant cette maladie sera directement liée à l'entrée de l'index donnant la position du concept dans le document. Dans le cas d'un document XML ou d'une base de données, on trouvera aussi une entrée dans l'index donnant la localisation de ce concept dans la structure de cette source de donnée.

Dans le cas d'un document (semi-)structuré, le concept peut être représenté par l'une de ses structures. Par exemple, le concept « pathologie » représenté par une table de base de données, destinée à contenir un ensemble de pathologie, sera lié à la description de cette table. Il est aussi possible que le concept a trouvé soit une instance d'un autre concept représenté par une des structures de la source de donnée. Imaginons que la recherche porte sur la pneumonie et qu'une base de données (non indexée) ait une table représentant le concept de pathologie. Il est donc possible que cette table contienne une ligne traitant de la pneumonie. En supposant que la correspondance entre le concept « Pathologie » et la table soit enregistrée dans la métabase, il est possible de la retrouver à partir du concept « Pneumonie ». Pour cela, on cherche les concepts subsumant « Pneumonie » ce qui permet de trouver « Pathologie » et finalement on trouve la table susceptible de contenir l'enregistrement relatif à cette maladie.

Plus formellement, cette étape consiste à identifier l'ensemble des sources de données potentiellement pertinentes et pour chacune, l'ensemble de leurs composants représentants, référençant ou susceptibles de l'avoir comme instance. Pour un concept donné, la recherche des composants le représentant ou le référençant se fait simplement grâce aux liens « référence » et « représente » qui les mettent en correspondance. La recherche des composants susceptibles de l'avoir pour instance, se fait en explorant la hiérarchie de subsomption du concept c.

Lorsqu'on a localisé les sources de données pertinentes et leurs composants, la suite du processus de recherche dépend du type de source de donnée.

Elle consiste à :

- Calculer un indice de pertinence et déterminer la granularité de l'unité d'information à retourner
- L'accès à l'unité d'information pertinente auprès de la source de donnée (qui peut nécessiter de formuler la requête)
- Exécuter la requête.

L'unité d'information doit être à la fois la plus spécifique et tout en répondant de manière exhaustive au besoin en information.

Le chapitre suivant est consacré au cas des documents non structurés, le chapitre 3.7 à celui des documents semi-structurés et le chapitre 3.8 aux données structurées.

3.6. Interrogation de documents non structurés

Nous allons discuter ici des traitements spécifiques aux documents non structurés. Cette étape commence lorsque la source de données identifiées lors de l'étape de localisation est en fait un document « plat ». On a donc localisé (faire référence au chapitre consacré à la modélisation des documents non structurés) :

- l'entité d'information (« info entity ») représentant le document
- les entités d'information donnant la position, dans le document original, des concepts pertinents par rapport à la requête

Le calcul de la pertinence d'une requête par rapport à un document plat est réalisable en utilisant un des modèles vus au chapitre 1.1.6, dont le modèle vectoriel qui donne les meilleurs résultats. En ce qui concerne la détection de l'unité d'information pertinente, l'absence de structure fait qu'il est très difficile de développer un système capable de retourner une granule plus fine que le document entier de manière efficace. L'unité d'information sera donc le document entier.

L'accès à l'unité d'information c'est-à-dire le document est aussi très simple dans ce cas de figure. On le récupère simplement à l'adresse contenue dans le champ « Location » de l'«Info entity » représentant le document. Afin d'aider l'utilisateur, il est possible de mettre en surbrillance dans le document, les termes correspondants aux concepts pertinents.

3.7. Interrogation de documents semi-structurés

L'interrogation d'un document semi-structuré se fait à partir des informations sur son modèle et la représentation de son contenu. Grâce aux correspondances entre les concepts de l'ontologie et les descriptions de documents non structurés, on dispose des informations suivantes, toutes pertinentes par rapport à la requête :

- Un ensemble de déclarations d'éléments
- Un ensemble d'entrée dans l'index de document avec pour chacune :
    - L'instance de document
    - La déclaration de l'élément
    - Eventuellement, les valeurs des attributs permettant d'identifier l'élément dans le document

Le calcul de la pertinence est affecté par la structure et par l'imbrication des éléments. De nombreux modèles sont détaillés dans la littérature comme nous l'avons vu dans l'état de l'art consacré à la RI structurées, chapitre 1.2.3.

Un document XML possède une structure arborescente, la sélection du bon niveau de granularité consiste à choisir le nœud qui soit le plus exhaustif et le plus spécifique pour répondre à la requête. Un tel nœud correspond à l'élément qui englobe tous les éléments trouvés à partir de l'index ou des déclarations d'éléments. Si le contenu du nœud sélectionné est trop spécifique, on laissera à l'utilisateur la possibilité d'afficher du contenu plus général en remontant dans l'arborescence.

Lorsque le système a sélectionné le nœud à retourner, il est possible de récupérer son contenu en interrogeant le document XML à l'aide d'une requête XPath. Cette requête permet d'exprimer le chemin absolu de l'élément dans le document. On peut la construire à partir de la déclaration de l'élément. Le problème est qu'il peut y avoir plusieurs éléments de ce type pour un même chemin. Si l'élément a été trouvé à partir d'une entrée dans l'index, peut comme on l'a vu au chapitre 2.4, y retourner les valeurs de ses attributs permettant d'identifier l'élément de manière unique.

3.8. Interrogation d'une base de données

L'interrogation d'une base de données se fait à partir d'informations sur sa structure décrite dans le schéma logique, et le contenu indexé. On considère que le plus petit niveau de granularité dans une base de données est la ligne, on n'ira donc pas jusqu'à retourner une seule colonne. Via les concepts identifiés dans l'ontologie, on a pu sélectionner :

- Un ensemble de tables
- Un ensemble d'entrées dans l'index avec pour chacune :
    - La table représentant le concept
    - Eventuellement, les valeurs des colonnes permettant d'identifier la ligne dans la table

L'interrogation d'une base de données à partir d'un système de recherche d'informations est un cas très complexe. En effet, la communauté des bases de données et la communauté liée à la recherche d'information utilisent des approches très différentes. La première différence porte sur la formulation de la requête qui dans le cas des BDs nécessite d'en connaître la structure pour préciser la forme du résultat et les conditions sur les valeurs de retour. En recherche d'informations, la requête porte sur le contenu sans considération pour la structure. Une autre différence est liée à la manière d'interpréter la requête, booléenne pour les langages de base de donnée et basée sur des mesures de pertinences dans les SRI. Notre système étant orienté

recherche d'informations, c'est lui qui va devoir déterminer la granularité et la forme du résultat à retourner.

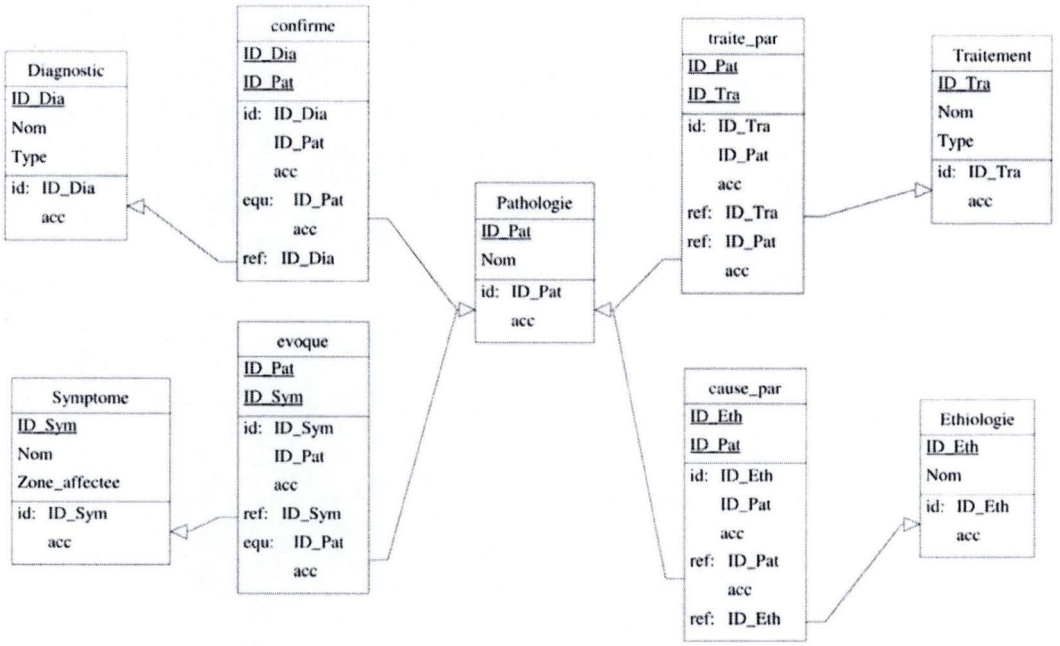


Figure 48 Exemple de base de données

Afin de pouvoir interroger la base de données, le système devra donc être capable de formuler la requête dans un langage adéquat comme SQL. Ce procédé dépasse le cadre de ce mémoire, mais nous allons le décrire de manière très abstraite à travers un petit exemple. Considérons le cas de la recherche d'informations portant sur les concepts « pneumonie » et « traitement» dans la base de données modélisée à la figure ci-dessus. Imaginons que le concept « traitement» nous permette de trouver la description de la table du même nom ; le concept « pneumonie » nous permet de trouver une entrée dans l'index concernant la table « Pathologie » dont la valeur de l'attribut « ID_Pat» est 3.

Le choix du niveau de granularité n'est pas un réel problème. Lorsque la recherche porte sur plusieurs tables, le niveau granularité sera l'union des colonnes de ses tables. Ici, on affichera les colonnes des tables « traitement» et « pathologie » à l'exception des colonnes techniques (identifiant et clef étrangère). On considère aussi que les tables référencées par une des tables sélectionnées qui ne référence et ne sont référencée par aucune autre décrivent le même concept. En plus de ces colonnes, il serait utile d'afficher des liens vers les autres tables référencées ou qui référencent celles qui sont affichées sous forme d'hyperlien. Cela permettrait à l'utilisateur de pouvoir naviguer dans la base de données.

Pathologie				Traitement	
Libellé	Etiologie	Symptômes	Diagnostic	Libellé	Type
Pneumonie	Liste	Liste	Liste	Antibiotique	Médicamenteux
Pneumonie	Liste	Liste	Liste	Antipyrétique	Médicamenteux
Pneumonie	Liste	Liste	Liste	Hydratation	Non-médicamenteux

Tableau 4 Exemple d'affichage pour la réponse d'un BD à la requête 'pneumonie traitement'

Le résultat attendu par l'utilisateur est obtenu en effectuant une jointure entre les tables pathologie et traitement. Pour cela, il faut trouver les colonnes qui lient les enregistrements de ces deux tables, via le schéma logique. On spécifiera aussi que la valeur de l'attribut « id » pour la table pathologie doit être égal à 3.

Le principal problème pour interroger ce type de sources de donnée est de trouver un modèle de calcul de la pertinence applicable. Les modèles utilisés pour les documents semi-structurés sont un bon point de départ, mais devront être adaptés. En effet, ces modèles utilisent la redondance des informations pour calculer l'importance d'un terme (ici une donnée) ; or en base de données, on cherche justement à éviter les redondances.

Conclusion

Dans ce chapitre, nous avons vu comment exploiter les descriptions de documents afin de permettre à un utilisateur d'y trouver de l'information.

Nous avons d'abord vu les différentes façons pour cet utilisateur d'exprimer son besoin en information. La solution la plus courante étant de l'exprimer via une requête composée de mots-clefs ; la plus simple étant de lui permettre de sélectionner directement les concepts de l'ontologie grâce à un mécanisme d'autocomplétion. Lorsque la requête est exprimée sous forme de mots-clefs, nous avons vu différentes techniques permettant de trouver les concepts visés par les termes composant la requête.

On s'est ensuite intéressé à l'enrichissement de la requête. On profite des relations sémantiques et hiérarchiques entre les concepts de l'ontologie pour permettre à l'utilisateur de repréciser son besoin en information. Toutefois, comme on l'a vu, il ne faudra pas oublier de recalculer la pondération des concepts dans la requête.

Une fois les concepts de l'ontologie visé par la requête utilisateur identifiés, il est facile de retrouver les granules pertinentes dans les descriptions de documents. On peut alors en fonction du type de document aller interroger la source de donnée contenant ce dernier.

Le cas des documents texte ne pose pas de problèmes, une fois sa description identifiée, on connaît son adresse et il est aisé de le récupérer. Le cas des documents XML est un peu plus complexe, mais pas insurmontable, la description de leurs structures et les index permettent de construire une requête de type XPath pour récupérer l'information pertinente. Le cas des bases de données soulève beaucoup plus de questions, notamment sur la façon de reconstruire la requête permettant de récupérer l'information dans les tables jugées pertinentes.

Chapitre 4

Implémentation d'une interface pour la métabase

4.1. Introduction

Nous avons défini une plateforme logicielle permettant d'insérer des descriptions de documents dans la métabase. Cela nous a permis de concrétiser notre approche de modélisation des données hétérogènes présentée au Chapitre 2. Le but n'est pas de créer ces descriptions, mais simplement d'insérer les index et modèles de données (Schéma conceptuel, Schéma logique, DTD, ...) qu'elles agrègent.

La plateforme doit permettre d'insérer les artefacts suivants :

- Description de documents « plats », c'est à dire les index représentant le contenu des documents (voir section 2.4)
- Modèles XML, c'est-à-dire des XML Schema et DTD (voir section 2.5)
- Schémas conceptuels et logiques, décrivant la structure d'une Base de données (voir section 2.6)
- Ontologies (voir section 0)

Le processus d'insertion d'une description de document dans la le modèle de la métabase se fait en deux étapes. La première consiste à **transformer** cette description, respectant un certain modèle et utilisant les constructions qui y sont définie, dans le modèle de la métabase. Des règles de transformations ont d'ailleurs étés définies au Chapitre 2 pour transformer un index, une DTD, un schéma conceptuel ERA, un schéma logique en une description conforme au modèle de la métabase. Une fois la description exprimée dans le modèle de la métabase, on peut l'y **importer**, c'est la deuxième étape. Afin de bien séparer ces deux étapes, on passera par un **fichier XML d'échange de métadonnée**, permettant d'exprimer une description conforme au modèle de la métabase dans un fichier XML. La figure ci-dessous illustre le processus d'insertion d'une DTD dans la métabase.

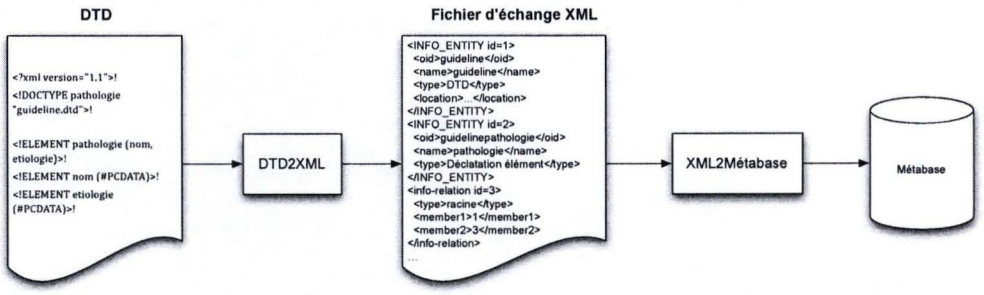


Figure 49 Processus d'insertion d'une description dans la métabase

Le programme « DTD2XML » est chargé de lire une DTD et l'exprimer en une description conforme au modèle de la métabase, en appliquant les règles de transformations définies au chapitre 2.5.1. Le résultat est enregistré dans un fichier d'échange de métadonnée. Ce fichier est lu par le programme « XML2Métabase » qui permet d'insérer une description contenue dans un fichier d'échange dans la base de donnée. Ici, le programme « XML2Métabase » insère n'importe quel description tant qu'elle respecte les constructions définies par le modèle de la métabase ; On peut imaginer un programme qui vérifie aussi la sémantique des descriptions fournies en entrée ; Dans notre cas, on aurait pu vérifier que la descriptions soit conforme au modèle des DTD.

Afin de réaliser au mieux cette tâche, nous avons défini une architecture qui est détaillée dans le chapitre suivant. Nous passerons ensuite à un petit aperçu des technologies employées pour réaliser cette architecture. Finalement nous parlerons d'un prototype réalisé lors d'un stage au Cetic. Il s'agit d'une plateforme Web permettant d'insérer les descriptions de document.

4.2. Architecture

Nous avons défini une architecture pour faciliter la réalisation des programmes de transformation et une autre pour les programmes d'importation.

On a besoin d'autant de programme de transformation qu'il existe de formalisme pour décrire un type de document. Par exemple pour décrire la structure d'un document XML : on aura besoin d'un programme pour transformer une DTD en une description conforme au modèle de la métabase ; et un autre programme prenant en charge les XML Schémas. Comme ces programmes n'insèrent pas le résultat directement dans la méta-base mais dans un fichier XML d'échange de métadonnée, ils utilisent une API permettant de ces fichiers qui doivent se conformer au schéma de la métabase.

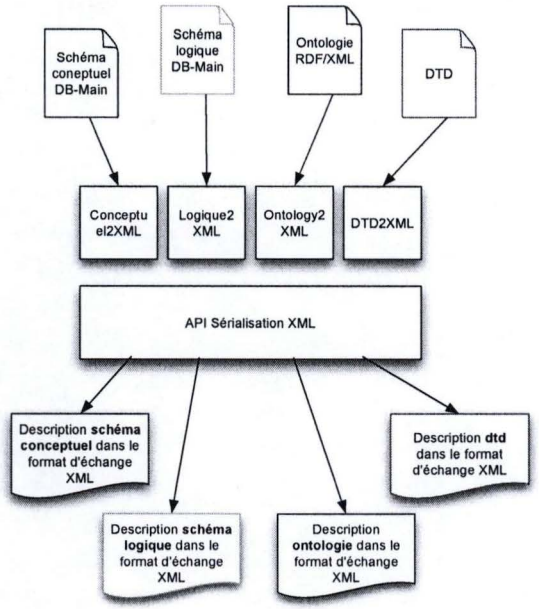


Figure 50 Architecture des programmes de transformations

La figure ci-dessus illustre l'architecture des programmes de transformations de descriptions. La figure ci-dessous donne un exemple de fichier XML d'échange, contenant une « Info Unit ».

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
 <INFO-UNIT id="1">
  <OId>ExempleInfoUnit</OId>
  <ObjectName> ExempleInfoUnit </ObjectName>
  <Name>MonInfoUnit</Name>
  <TypeParameters>root</TypeParameters>
  <Type>Exemple</Type>
  <Medium>File</Medium>
 </INFO-UNIT>
</root>
```

Figure 51 Exemple de fichier d'échange XML

Le contenu du fichier d'échange est ensuite stocké dans la métabase via un programme d'importation. On trouve deux types de programmes d'importation :

- Ceux de la première catégorie, dont fait partie « XML2Metabase », importe le contenu d'un fichier XML dans la métabase tant que sa structure est conforme au modèle de la métabase ;

- Ceux de la seconde catégorie, dont fait partie « Ontologie2Metabase », s'intéressent a la sémantique de la description contenue dans le fichier XML. Ici, « Ontoloigie2Metabase » vérifierait que le contenu du fichier fournis en entrée décrive bien une ontologie et soit conforme à un certain modèle d'Ontologie.

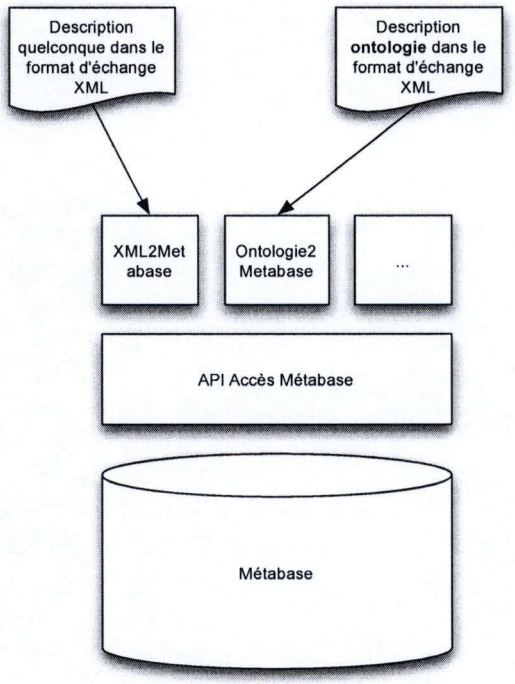


Figure 52 Architecture programmes d'importation

La figure ci-dessus représente l'architecture des programmes d'importation. Ces programmes utilisent l'API d'accès à la métabase créée par le générateur d'API Gisele. Cette API permet de réaliser les accès à la métabase à partir d'objet Java.

4.3. La plateforme d'administration de la métabase

Une plateforme Web a été développée pour rassembler les programmes de transformations et d'importations de descriptions dans la métabase. Ces programmes prennent alors la forme de plugin. L'architecture de la plateforme est décrite dans le schéma ci-dessous.

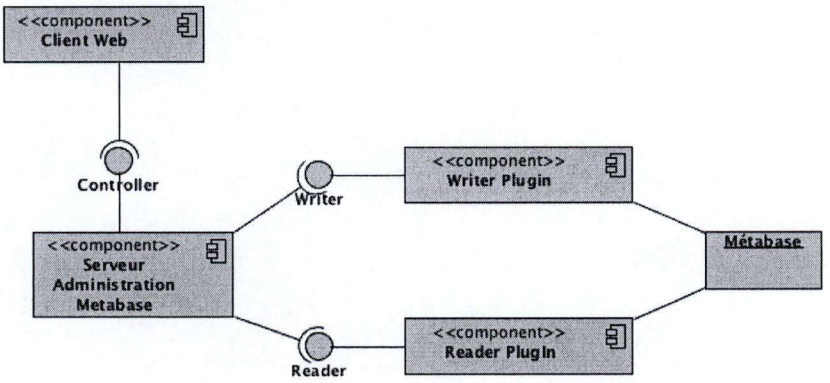


Figure 53 Composants de la plateforme Web

Un « Writer Plugin » est un programme chargé de transformer et importer un certain type de description (schéma conceptuel, schéma logique, ontologie, ...) dans la métabase. Un « Reader Plugin » est un programme permettant de visualiser un certain type de description stocké dans la base. On a défini une interface « Writer » et « Reader » spécifiant les méthodes que les plugins doivent implémenter pour que le Serveur d'Administration de la Métabase (composant « Serveur Administration Metabase ») puisse interagir avec eux.

Le composant « Serveur Administration Metabase » est un serveur Web, implémenté en JSP, contrôlé depuis une interface Web (composant « Client Web »). Lorsqu'un utilisateur souhaite ajouter un certain type de description, il délègue la tâche au « Writer Plugin » approprié. Lorsqu'il doit afficher la liste des descriptions ou une description particulière, il transmet la tâche au « Reader Plugin » capable de lire ce type de description.

Le cahier des charges qui a précédé le développement de cette plateforme est mis à disposition en annexe 5.2.

4.3.1. Technologies utilisée

Java

Java est, un langage de programmation orienté objet, utilisé pour le développement des différents outils. Son choix s'est imposé en raison de sa popularité et afin de pouvoir utiliser le plugin DB-Main et l'API d'accès à la base de donnée.

JSP

JSP, pour JavaServer Page est une librairie, incluse dans la version Entreprise de Java (JEE), utilisé pour créer des applications Web Dynamique. Il permet notamment de

faire appel à du code Java, qui sera exécuté coté serveur, depuis une page Web écrite en JSP.

JGraph

JGraph[11] est une librairie Java open source qui permets de mettre en page et dessiner des graphes. Elle a été utilisée pour pouvoir afficher un graphique représentant les schémas logiques et conceptuels stockés dans la métabase.

DB-Main

DB-Main[12] est un logiciel de modélisation de donnée. Il est utilisé pour créer des schéma conceptuel, schéma logique, diagramme d'activité. Il inclus des outils de transformations de schéma. Nous l'avons choisi, car il est très populaire au FUNDP, d'où il est originaire[13]. Nous avons utilisé la librairie normalement conçue pour créer des plugins DB-Main, afin d'exploiter ses capacités de lecture de schéma conceptuel et logique, depuis nos applications ; Ces schéma ayant été créé à partir de DB-Main.

Jena

Jena[14] est une librairie permettant de d'utiliser les technologies du Web Sémantique en Java. Il nous a été utile pour lire des ontologies RDF et OWL afin de les encoder dans la métabase.

Générateur d'API GISELE

Le générateur d'API, développé dans le cadre du projet GISELE (voir chapitre 2.2), permet de générer les méthodes d'accès à la base de données. La génération du code Java s'opère à partir du schéma conceptuel et logique de la base de données. En enregistrant les liens qui existent entre schéma conceptuel et logique, il génère des classes Java conformes au modèle conceptuel qui permettent d'accéder à la base de données ; elle même générée à partir du schéma logique.

4.3.2. Plugins pour les schémas conceptuels et logiques

Afin de supporter les schémas conceptuels, nous avons développés deux plugins, permettant respectivement d'insérer un schéma logique et un schéma conceptuel stocké dans un fichier lun[15]. Un autre programme avait été développé pour insérer un

---

[11] JGraph : http://www.jgraph.com/.
[12] DB-Main : http://www.db-main.eu.
[13] Histoire de DB-Main : http://www.db-main.eu/?q=fr/node/26.
[14] Jena : http://jena.sourceforge.net/.
[15] lun : Format des fichiers créés par DB-Main

schéma logique, le schéma conceptuel dont il dérive et les liens entre ces deux schémas dans la méta-base mais il n'a pas été intégré à la plateforme Web par manque de temps. La librairie Java de DB-Main est utilisée pour naviguer dans le schéma et chaque élément de celui ci est transformé dans une entité de la métabase, selon les règles définie au chapitre 2.6.1 pour un schéma conceptuel et 2.6.2 pour un schéma logique.

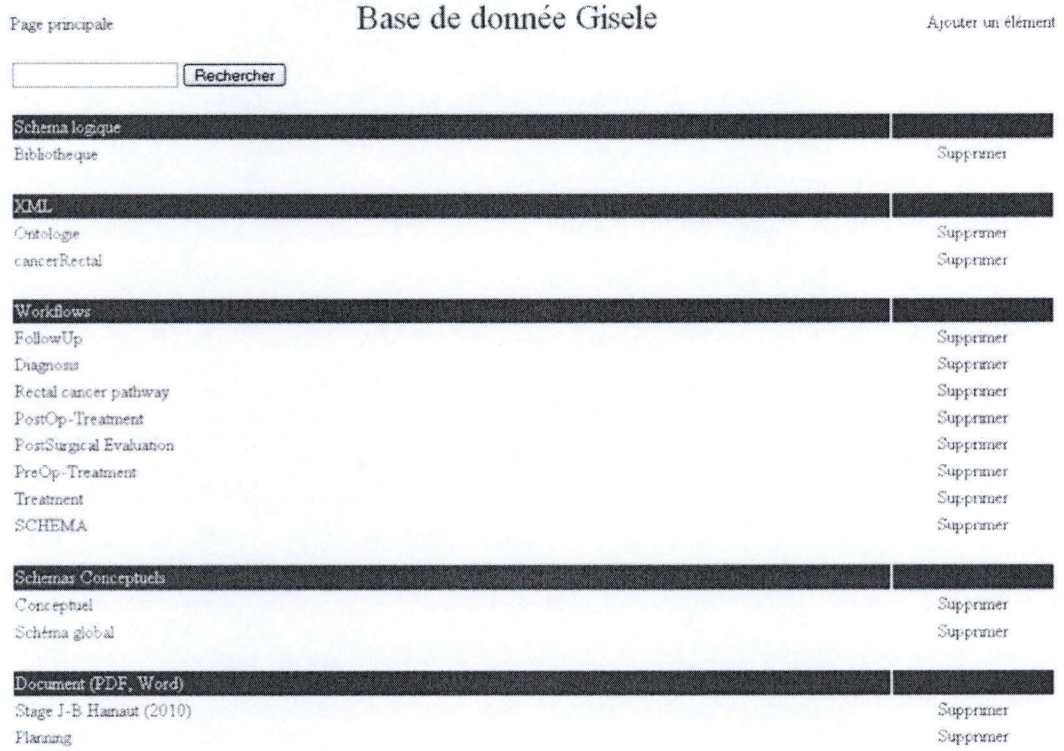
Afin de pouvoir représenter graphiquement les schémas conceptuels et logiques stockés dans la métabase, nous avons utilisé le JGraph.

4.3.3. Plugin pour les ontologies

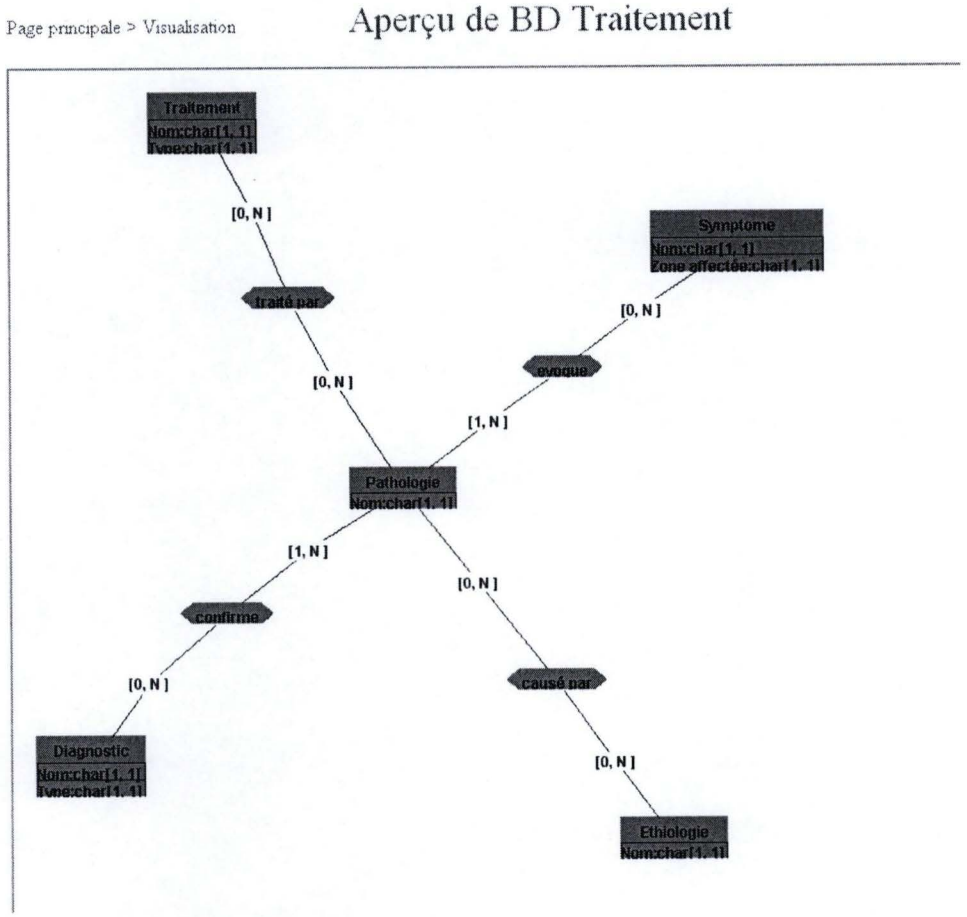
Ce plugin permet d'encoder une ontologie dans la métabase. La bibliothèque Jena est utilisée pour naviguer dans l'ontologie source, écrite RDF ou OWL. Chaque entité de l'ontologie source est alors transformée en une entité propre à la métabase, selon les règles de transformations définie au chapitre 0.

4.3.4. Utilisation

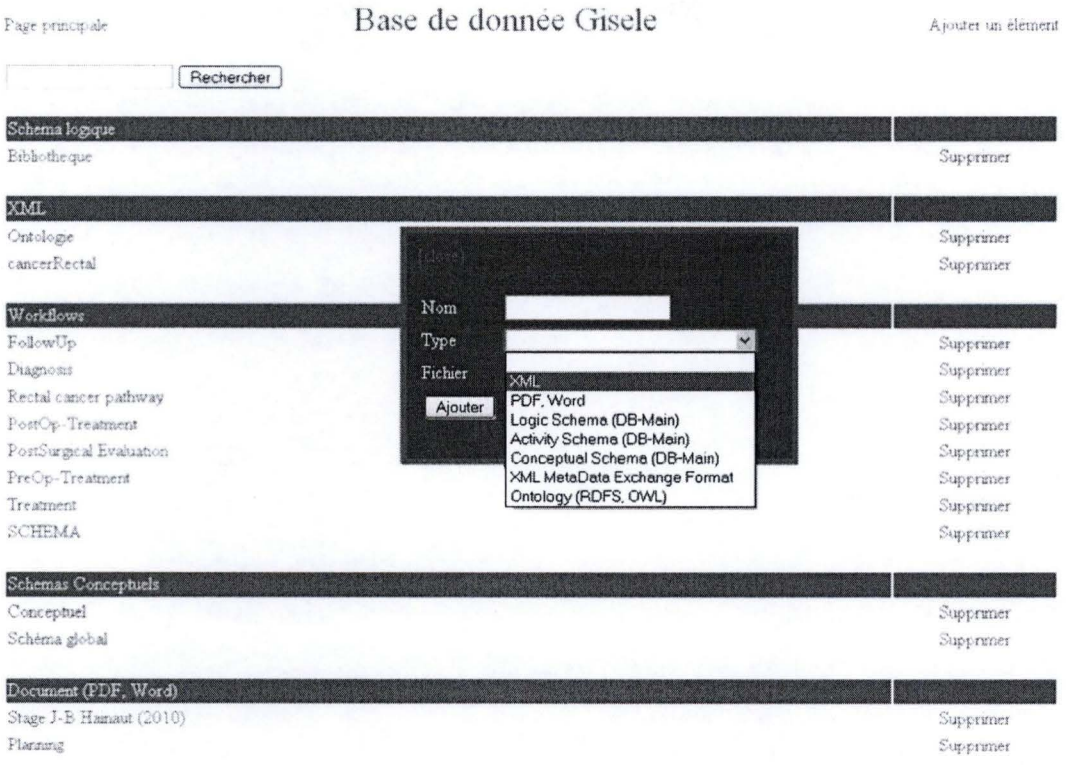
La page d'accueil de l'interface Web présente la liste des descriptions stockées dans la métabase. Celle-ci sont classée par type : Schéma logique, Schéma conceptuel, Ontologie, ... Pour information, les types de descriptions XML, Workflows, Document PDF et Word sortent du cadre de ce mémoire.



L'utilisateur peut visualiser une description en cliquant sur son nom. Il peut aussi la supprimer en cliquant sur le lien « Supprimer » en regard de son nom.

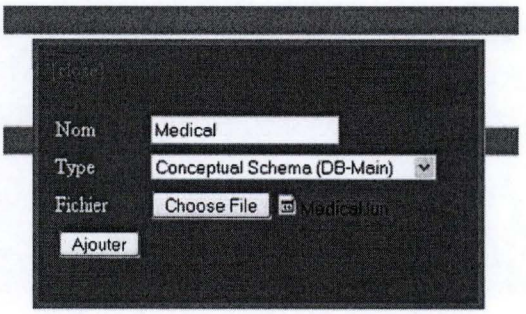


La capture ci-dessus présente la page permettant de visualiser une description. Dans ce cas, il s'agit du schéma conceptuel d'une base de donnée.

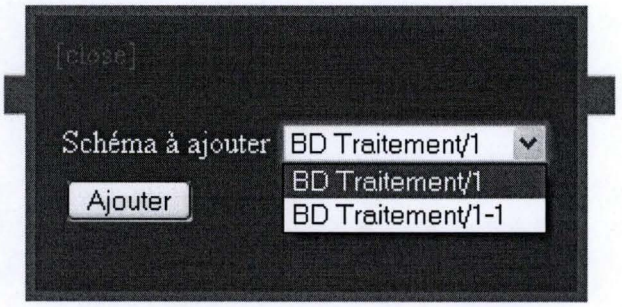


Une fois de retour sur la page d'accueil, il est possible d'ajouter une nouvelle description, en cliquant sur le lien « ajouter un élément » en haut à droite. Un formulaire apparaît alors en surimpression :

1. L'utilisateur choisis dans la liste, le type de description à ajouter
2. Il clique ensuite sur « Choisir un fichier » pour sélectionner le fichier contenant la description à ajouter. Une fois le fichier sélectionné, il valide ce qui le ramène au formulaire.
3. Comme on le voit sur l'image ci-dessous, le champ nom est alors pré-rempli, sur base du nom du fichier. Il peut si il le souhaite le modifier.
4. Finalement, il clic sur ajouter.



5. Si le fichier contient plusieurs descriptions différentes, comme dans le cas d'un fichier DB-Main composés de plusieurs schémas, un nouveau formulaire invite l'utilisateur à sélectionner celui qu'il souhaite ajouter. Après avoir cliquer sur « ajouter », le schéma est enregistré dans la métabase.



Conclusion et perspectives

Dans ce mémoire, nous avons étudié la représentation, la gestion et l'exploitation de description de documents non structurés, de documents semi-structurés et de données structurées. Nous avons utilisé les documents textes pour représenter les documents non structurés, les documents XML pour les documents semi-structurés et les bases de données.

Nous avons d'abord proposé une méthode pour la modélisation des descriptions de ces documents. Pour cela, nous avons utilisé la partie « Information » de la base de données développée dans le cadre du projet GISELE, que nous avons appelée métabase. Nous avons ensuite abordé les différentes approches, pour encoder ces descriptions dans une base de données et pour développer le schéma d'une telle base de données. Ensuite, nous avons détaillé les transformations permettant d'exprimer les descriptions de documents textes, documents XML et base de données dans le modèle de la métabase. Cette étape est nécessaire pour pouvoir y importer ces descriptions. Nous avons aussi défini les transformations nécessaires à l'encodage d'ontologies RDFS dans la métabase. Ensuite, nous nous sommes intéressés à la représentation des correspondances entre descriptions dans la métabase. Deux types de correspondances étaient particulièrement utiles : les correspondances verticales entre schéma conceptuel et schéma logique et les correspondances horizontales entre l'ontologie du domaine et les descriptions de documents.

La seconde partie du mémoire est consacrée à l'exploitation de ces descriptions afin de rechercher des informations dans les documents décrits. Nous avons d'abord discuté les différentes possibilités d'expression du besoin utilisateur. La solution la plus répandue est d'exprimer la requête sous forme d'une liste de mots clefs ; la plus simple est de référencer directement les concepts de l'ontologie, soit en utilisant un mécanisme d'autocomplétion pour aider l'utilisateur à sélectionner les concepts appropriés de l'ontologie, via leurs descriptions en langage naturel, contenus dans les labels associés à chaque concept, soit via une représentation graphique de l'ontologie. Dans le cas où le choix se porte sur une requête constituée de mots-clefs, nous avons mis en avant les techniques permettant d'identifier les concepts visés par ces mots. Une fois les concepts de l'ontologie intéressant l'utilisateur identifiés, il est aisé de trouvés les granules pertinentes dans les descriptions de documents. Ensuite sont abordées les techniques permettant de récupérer l'information dans les documents ; ces techniques sont spécifiques à chaque type de documents. Le cas de la récupération des documents « plats » sur base de leur description est trivial. Le cas des documents XML est un peu plus complexe, mais réalisable, en utilisant une requête de type XPath qui peut être construite sur base des informations contenues dans sa description. Pour ce

qui concerne les bases de données, j'ai seulement abordé le fonctionnement du système qui permettra de reconstruire la requête permettant de récupérer les données pertinentes.

Finalement, nous avons envisagé le fonctionnement de la plateforme Web destinée à faciliter l'encodage des descriptions de documents dans la métabase. Cette plateforme permet d'ajouter des plugins pour l'importation de nouveau type de descriptions.

Beaucoup de travail doit encore être fait pour permettre à un utilisateur de rechercher de l'information, de manière transparente, dans des données hétérogènes. Dans ce mémoire, nous avons émis l'hypothèse que les descriptions de documents existaient et que les correspondances avec l'ontologie du domaine étaient connues. Il faut donc encore trouver des méthodes d'indexation efficaces, avec l'ontologie du domaine, pour les différents types de document. Il faut également définir un modèle pour le calcul de la pertinence et le choix du bon niveau de granularité de l'information à renvoyer. Bien que de nombreux modèles existent pour évaluer la pertinence d'un document XML par rapport à une requête, il n'existe pas encore un tel modèle pour les bases de données. Il faut enfin développer une technique permettant construire une requête pour interroger les bases de données. Cette requête devra être construite sur base de tables jugées pertinentes par rapport à la requête.

L'opportunité de disposer d'un système permettant de rechercher de l'information dans des sources de données hétérogènes, et cela de façon transparente pour l'utilisateur, est particulièrement attractive. Néanmoins des années de travail seront nécessaires avant qu'un tel système voie le jour.

Glossaire

Concept

Dans ce mémoire, ce terme fait référence aux concepts d'une Ontologie.

Description

Nous utilisons le terme description pour faire référence à la représentation d'un document. Celle-ci étant constituée d'un index représentant son contenu ; et dans le cas d'un document structuré, d'un modèle décrivant sa structure.

Document non-structuré

Un document non-structuré, aussi appelé document « plat », est un document dépourvu de structure logique.

Document semi-structuré

Un document semi-structuré est un document dont le contenu est structuré, généralement à l'aide de balise. XML est un exemple de langage permettant de définir la structure logique des documents. On utilise le terme semi-structuré, plutôt que structuré, car ses granules contiennent du texte plutôt que des données typées.

Donnée structurée

Une donnée structurée est une donnée qui possède un type bien spécifique.

Enrichissement de requête

Processus permettant de compléter la requête de l'utilisateur afin de le guider dans sa recherche et ainsi améliorer les résultats de la recherche.

Fichier d'échange de métadonnée

Il s'agit d'un fichier XML utilisé comme moyen de stockage intermédiaire par les programmes utilisés pour la gestion de la métabase. Le contenu de se fichier doit se conformer au modèle de la métabase.

Granule

Dans ce mémoire, le terme granule, désigne la plus petite partie de la structure logique d'un document.

Méta-modèle

Un métamodèle est un modèle décrivant un modèle.

Modèle

Dans ce mémoire, ce terme fait référence à un modèle de donnée qui décrit comment les données sont structurées.

Ontologie

Une ontologie informatique est « une spécification explicite et formelle d'une conceptualisation partagée ». Dans ce mémoire, nous utilisons une ontologie pour définir de manière formelle le domaine d'activité concerné par la Recherche d'Informations. L'ontologie permet contient les concepts du domaine ainsi que les relations sémantique qui existent entre eux.

Sémantique

La sémantique fait référence au sens d'un mots par opposition à sa syntaxe.

Unité d'information

Une unité d'information est la plus petite partie d'une information qui se suffit à elle même ; Il n'y a donc pas besoin de plus d'information pour pouvoir la comprendre.

Table des figures

Liste des tableaux

Bibliographie

An, Y., Borgida, A., & Mylopoulos, J. (2006). Discovering the Semantics of Relational Tables Through Mappings. *Journal on Data Semantics VII* , 1-32.

Anderson, J., & Pérez-Carballo, J. (2001). The nature of indexing: how humans and machines analyze messages and texts for retrieval: part II: machine indexing, and the allocation of human versus machine effort. *Information Processing and Management: an International Journal , 37* (2).

Andreasen, T., Bulskov, H., & Knappe, R. (2003). Similarity for Conceptual Querying. *Lecture notes in computer science* .

Assem, M. V., Menken, M. R., Schreiber, G., Wielemaker, J., & Wielinga, B. (2004). A Method for Converting Thesauri to RDF/OWL. *Proc. of the 3rd Int'l Semantic Web Conference , 3298*, 17-31.

Baziz, M., Boughanem, M., Aussenac-Gilles, N., & Chrisment, C. (2005). Semantic cores for representing documents in IR. *Proceeding SAC '05 Proceedings of the 2005 ACM symposium on Applied computing* .

Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The Semantic Web. *Scientific American Magazine* .

Borst, W. N. (1997). Construction of Engineering Ontologies for Knowledge Sharing and Reuse. *Thèse* .

Buckley, C., Allan, J., Salton, G., & Singhal, A. (1994). Automatic query expansion using SMART: TREC 3. *In Proceedings of the Third Text REtrieval Conference , 500* (225), pp. 69-80.

Carmel, D., Efraty, N., Landau, G. M., Maarek, Y. S., & Mass, Y. (2002). An Extension of the Vector Space Model for Querying XML Documents via XML Fragments. *XML and Information Retrieval workshop of SIGIR 2002* .

Conesa, J., & Olivé, A. (2006). A Method for Pruning Ontologies in the Development of Conceptual Schemas of Information Systems. *A Method for Pruning Ontologies in the Development of Conceptual Schemas of Information Systems , 3870*, 64-90.

Cui, H., Wen, J.-R., & Chua, T.-S. (2003). Hierarchical indexing and flexible element retrieval for structured document. *Proceeding ECIR'03 Proceedings of the 25th European conference on IR research* , 73-87.

Cui, H., Wen, J.-R., Nie, J.-Y., & Ma, W.-Y. (2002). Probabilistic query expansion using query logs. *Proceedings of the 11th international conference on World Wide Web* , pp. 325-332.

Fagan, J. L. (1989). The effectiveness of a nonsyntactic approach to automatic phrase indexing for document retrieval. . *Journal of the American Society for Information Science , 40* (2).

Fankam, C., Bellatreche, L., Hondjack, D., Ameur, Y. A., & Pierra, G. (2009). SISRO, conception de bases de données à partir d'ontologies de domaine. *Technique et Science Informatiques* , 1233-1261.

Fluit, C., Sabou, M., & Harmelen, F. V. (2003). Supporting User Tasks through Visualisation of Light-weight Ontologies. *Handbook on Ontologies in Information Systems* , 415-434.

Frakes, W. (1992). *Information Retrieval: Data Structures and Algorithms.* Prentice Hall.

Fuhr, N. (2001). XIRQL: A Query Language for Information Retrieval in XML Documents. *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval* , 172-180.

Gövert, N., Fuhr, N., Abolhassani, M., & Großjohann, K. (2003). Content-oriented XML retrieval with HyREX. *In Proceedings of INEX Workshop'2002* , 26-32.

Gandon, F. (2006). *Ontologies informatiques.* Récupéré sur Interstices: http://interstices.info/jcms/c_17672/ontologies-informatiques?part=0

Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition - Special issue: Current issues in knowledge modeling , 5* (2).

Guha, R., McCool, R., & Miller, E. (2003). Semantic Search. *Proceeding WWW '03 Proceedings of the 12th international conference on World Wide Web* .

Haav, H.-M., & Lubi, T.-L. (2001). A Survey of Concept-based Information Retrieval Tools on the Web. *Proc. of 5th East-European Conference , 2*, 29-41.

Hainaut, J.-L. (2009). *Bases de données : Concepts, utilisation et développement.* Dunod.

Hainaut, J.-L. (2003). *Ingénierie des bases de données.* FUNDP, Faculté d'Informatique, Namur.

Hainaut, J.-L., Brogneaux, A.-F., & Cleve, A. (2010). *Base de modèles pour les itinéraires de soins.* Université de Namur, Centre de recherche PReCISE / LIBD.

Hainaut, J.-L., Brogneaux, A.-F., & Cleve, A. (2010). *Healthcare Information System Modelling.* University of Namur, PReCISE Research center.

Horrocks, I., Patel-Schneider, P., & Harmelen, F. V. (2003). From SHIQ and RDF to OWL: The Making of a Web Ontology Language. *Journal of Web Semantics , 1* (1), 7-26.

Jean, S., Pierra, G., & Ait-Ameur, Y. (2007). DOMAIN ONTOLOGIES : A DATABASE-ORIENTED ANALYSIS. *Lecture Notes in Business Information Processing , 1*, 238-254.

Jones, K. S. (1971). *Automatic Keyword Classification for Information Retrieval.* Butterworths, London: Archon Books.

Lalmas, M. (1997). Dempster-Shafer's theory of evidence applied to structured documents: modelling uncertainty. *Proceedings of the 20th annual international ACM SIGIR conference on Research and development in information retrieval , 31*, 110-118.

Lawarrée, F. (2010). *Recherche de documents à partir d'ontologies de domaines .* FUNDP.

Lawarrée, F. (2010). *Recherche de documents à partir d'ontologies de domaines .* FUNDP, Namur.

Lee, Y. K., Yoo, S.-J., Yoon, K., & Berra, P. B. (1996). Index structures for structured documents. *Proceedings of the first ACM international conference on Digital libraries* , 91-99.

Lin, D. (1998). *An Information-Theoretic Definition of Similarity.* Winnipeg.

Luk, R. W., Leong, H., Dillon, T. S., Chan, A. T., Croft, W. B., & Allan, J. (2002, 02 22). A survey in indexing and searching XML documents. *Journal of the American Society for Information Science and Technology* , 415-437.

Maron, M., & Kuhns, J. (1960). On Relevance, Probabilistic Indexing and Information Retrieval. *Journal of the Association for Computing Machinery , 7* (3).

Mass, Y., Mandelbrod, M., Amitay, E., Carmel, D., Maarek, Y., & Soffer, A. (2002). JuruXML – an XML retrieval system at INEX'02. *In Proceedings of INEX Workshop'2002* , 73-80.

Meuss, H., & Schlieder, T. (2002). Querying and ranking XML documents. *Journal of the American Society for Information Science and Technology - XML , 53* (6), 489-5003.

Navigli, R., & Velardi, P. (2003). An Analysis of Ontology-based Query Expansion Strategies. *Workshop on Adaptive Text Extraction and Mining* .

Neches, R., & Al. (1991). Enabling technology for knowledge sharing. *AI Magazine , 12* (3), 36 - 56.

Ogilvie, P., & Callan, J. (2003). *Using Language Models for Flat Text Queries in XML Retrieval.* Carnegie Mellon University, Language Technologies Institute, Pittsburgh.

Resource Description Framework - Wikipedia. (2011, 04 30). Récupéré sur Wikipedia: http://fr.wikipedia.org/wiki/Resource_Description_Framework

RICŒUR, P. (2009). Ontologie. *Encyclopædia Universalis* .

Robertson, S. (1997). The probability ranking principle in IR.

Rocchio, J. (1971). Relevance feedback in information retrieval. *The SMART Retrieval System - Experiments in Automatic Document Processing* , pp. 313-323.

Salton, G., Wong, A., & Yang, C. S. (1975, Novembre). A vector space model for automatic indexing. *Communications of the ACM , 18* (11).

Sauvagnat, K., & Boughanem, M. (2009). *A la Recherche de noeuds informatids dans des corpus de documents XML.* IRIT- SIG, Toulouse.

Schlieder, T., & Meuss, H. (2002). Querying and Ranking XML Documents. *Journal of the American Society for Information Science and Technology , 53* (6), 489-503.

Singhal, A. (2001). Modern Information Retrieval: A Brief Overview. *IEEE Data Engineering Bulletin* .

Spaccapietra, S., Parent, C., Cullot, N., & Vangenot, C. (2004). On Using Conceptual Modeling for Ontologies. *Proceedings of the International Workshop on Intelligent Networked and Mobile Systems* , 22-23.

Studer, R., Benjamins, V. R., & Fensel, D. (1998). Knowledge Engineering: Principles and Methods.

Toussaint, Y. (2004). Extraction de connaissances à partir de textes structurés.

Uschold, M., & Grüninger, M. Ontologies: principles, methods, and applications. *Knowledge Engineering Review , 11* (2), 93-155.

Vallet, D., Fernández, M., & Castells, P. (2005). An Ontology-Based Information Retrieval Model. *The Semantic Web: Research and Applications* , 455-470.

Voorhees, E. M. (1994). Query expansion using lexical-semantic relations. *In Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval* , pp. 61-69.

W3C. (2004, 02 10). *RDF Primer.* Consulté le 08 06, 2011, sur W3C: http://www.w3.org/TR/rdf-primer/

W3C. (s.d.). *RDF Tutorial.* Récupéré sur W3C: http://www.w3schools.com/rdf/default.asp

W3C. (2004, 02 10). *RDF Vocabulary Description Language 1.0: RDF Schema.* Consulté le 07 10, 2011, sur W3C: http://www.w3.org/TR/rdf-schema/

W3C. (2008, 01 15). *SPARQL Query Language for RDF.* Consulté le 08 07, 2011, sur W3C.

W3C. (2010). *XML Schema.* Récupéré sur W3C: http://www.w3.org/XML/Schema

W3C. (s.d.). *XML Schema Tutorial.* Récupéré sur W3C Schools: http://www.w3schools.com/schema/

Wikipedia. (2010, 08 06). *Indexation automatique.* Consulté le 07 2011, sur Wikipedia: http://fr.wikipedia.org/wiki/Indexation_automatique

Wikipedia. (2011, 05 13). *Recherche d'information.* Consulté le 07 2011, sur Wikipedia: http://fr.wikipedia.org/wiki/Recherche_d'information

Zipf, G. (1972). *Human Behaviour and the Principle of Least Effort.* Hafner Pub. Co.

Chapitre 5

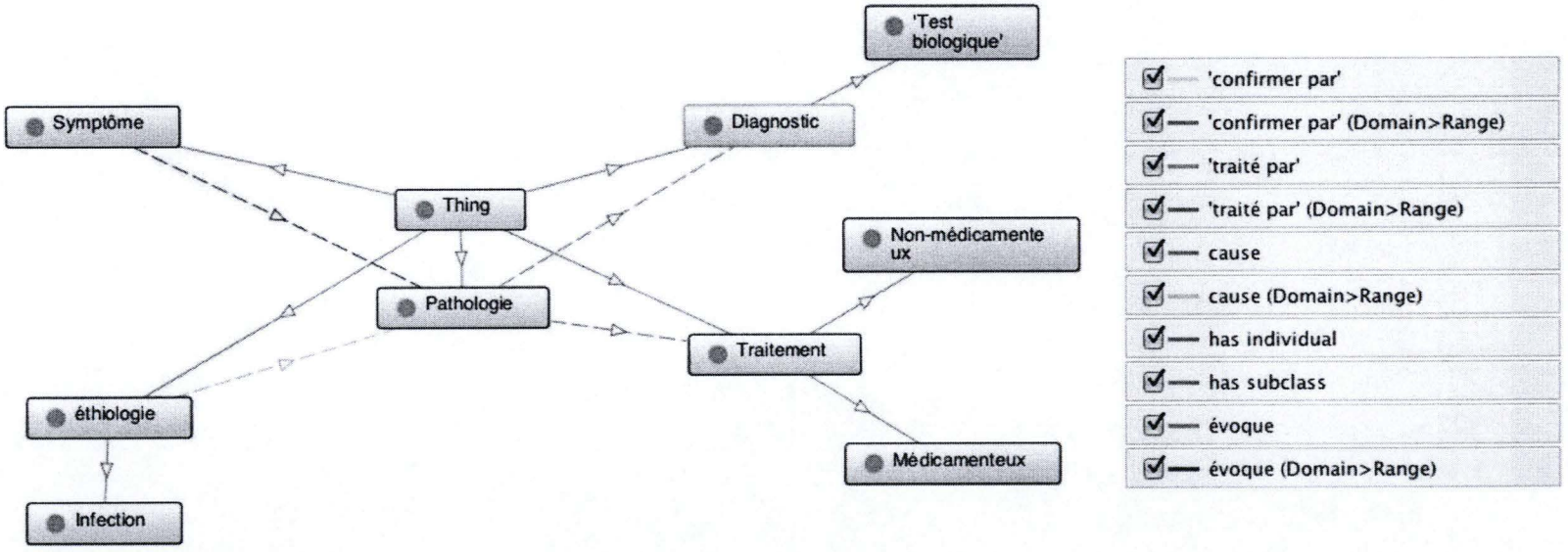
Annexe

5.1. Schémas

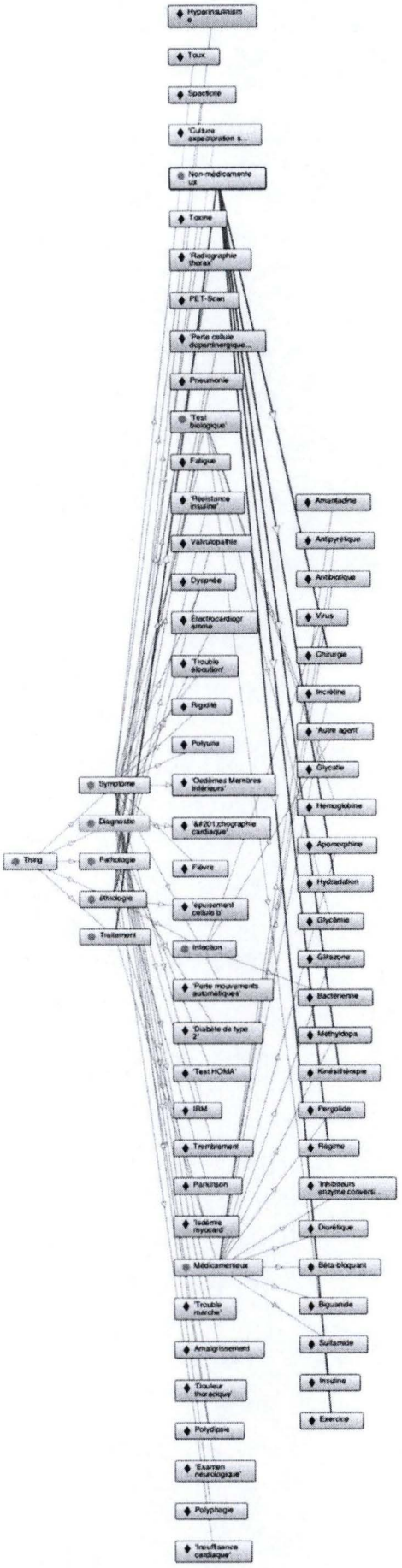
5.1.1. Ontologie du domaine

Le premier schéma présente les concepts du domaine utilisés pour les exemples dans ce mémoire. Le deuxième montre les individus appartenant à ces mêmes concepts.

Concept du domaine

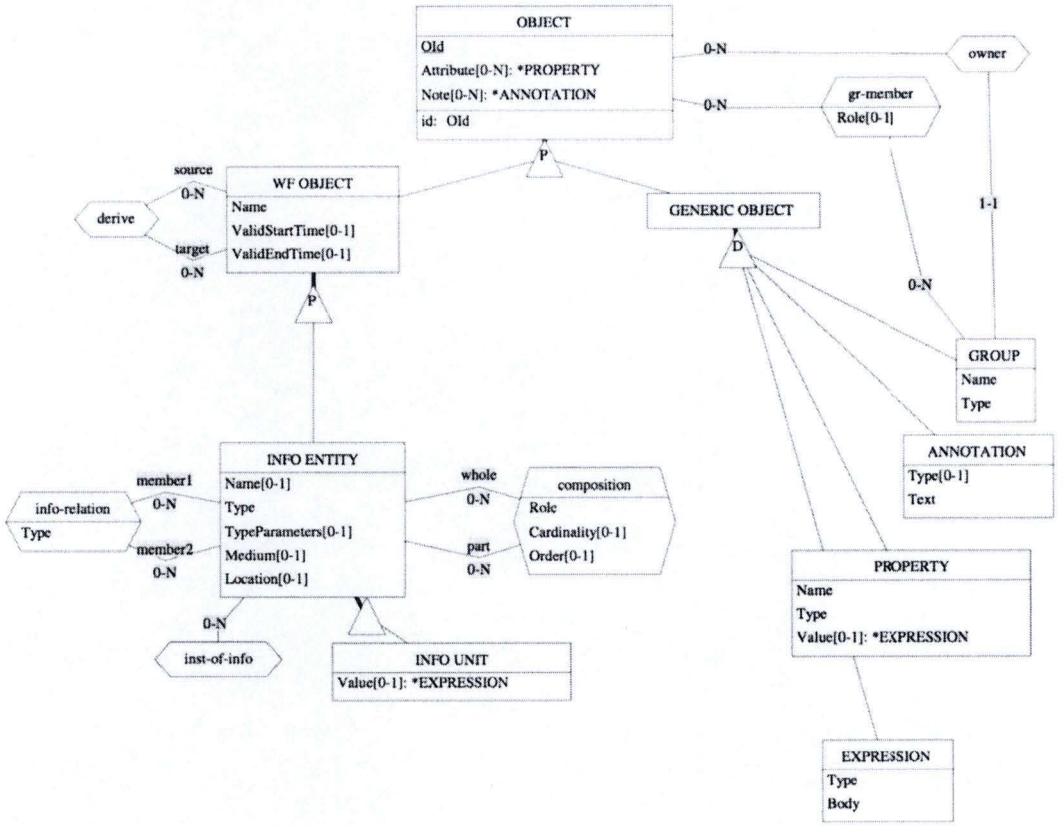

'Test biologique'
Symptôme
Diagnostic
Thing
Non-médicamente ux
Pathologie
Traitement
éthiologie
Médicamenteux
Infection
'confirmer par'
'confirmer par' (Domain>Range)
'traité par'
'traité par' (Domain>Range)
cause
cause (Domain>Range)
has individual
has subclass
évoque
évoque (Domain>Range)

Individus appartenant au concepts du domaine


Hyperinsulinisme
Toux
Spasticité
'Culture expectoration s...
Non-médicamenteux
Toxine
'Radiographie thorax'
PET-Scan
'Perte cellule dopaminergique...
Pneumonie
'Test biologique'
Fatigue
'Résistance insuline'
Valvulopathie
Dyspnée
'Electrocardiogramme
'Trouble élocution'
Rigidité
Polyurie
'Oedèmes Membres Inférieurs'
'Échographie cardiaque'
Fièvre
'épuisement cellule b'
Infection
'Perte mouvements automatiques'
'Diabète de type 2'
'Test HOMA'
IRM
Tremblement
Parkinson
'Isdème myocard
Médicamenteux
'Trouble marche'
Amaigrissement
'Douleur thoracique'
Polydipsie
'Examen neurologique'
Polyphagie
'Insuffisance cardiaque'
Symptôme
Diagnostic
Thing
Pathologie
étiologie
Traitement
Amantadine
Antipyrétique
Antibiotique
Virus
Chirurgie
Incrétine
'Autre agent'
Glycate
Hemoglobine
Apomorphine
Hydratation
Glycémie
Glitazone
Bactérienne
Methyldopa
Kinésithérapie
Pergolide
Régime
'Inhibiteurs enzyme conversi...
Diurétique
Bêta-bloquant
Biguanide
Sulfamide
Insuline
Exercice

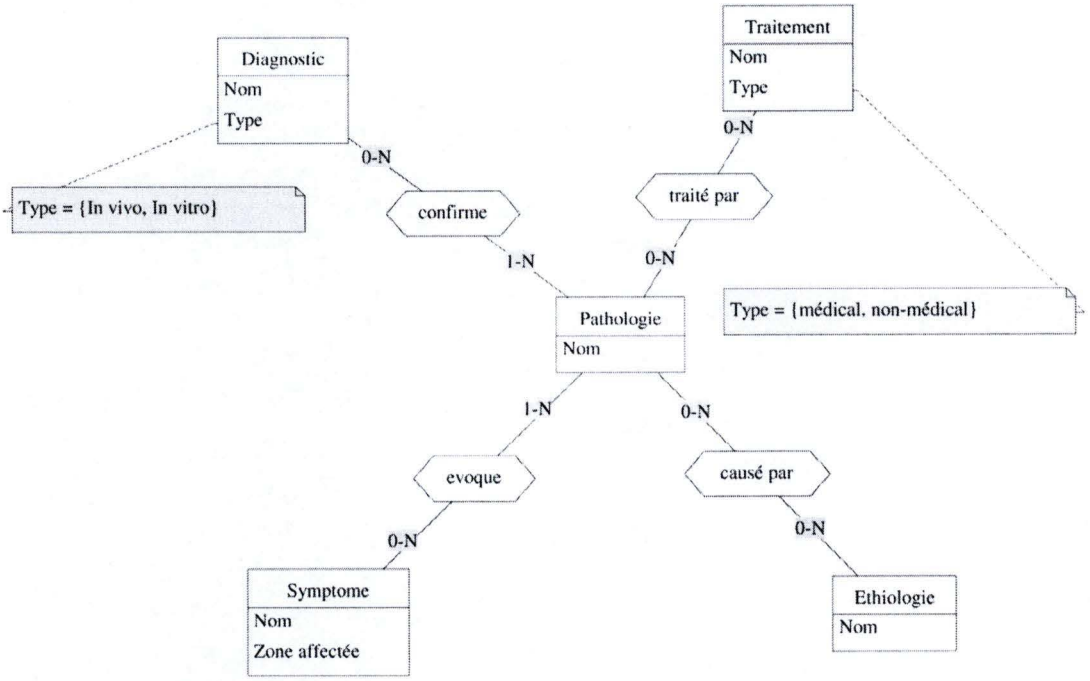
5.1.2. Schéma conceptuel de la métabase

Schéma conceptuel de la métabase, sous-schéma du schéma conceptuel de la base de donnée GISELE.



5.1.3. Exemple de schéma conceptuel de base de donnée

Le schéma conceptuel ci-dessous a servi pour les exemples relatifs aux base de données dans ce mémoire.



5.2. Interface Web pour la Base de Donnée Gisele : Analyse

5.2.1. Analyse des exigences

Introduction

Le projet s'inscrit dans le cadre du stockage de données hétérogène dans la base de donnée Gisele.

La base de donnée Gisele est destinée à accueillir tout documents utiles au domaine médical. Il doit être possible d'y stocker des schémas de base de donnée, des documents XML, PDF, ...

Objectif

L'objectif de ce projet est de créer une plateforme Web permettant d'interagir avec la base de donnée Gisele.

Cette plateforme composée d'une partie serveur et d'un client devra permettre de visualiser les documents présents dans la base de donnée. L'affichage de ses documents devra s'adapter en fonction de leur type c'est à dire que l'on ne présentera pas de la même manière un worflow et un document XML.

Il devra aussi être possible de rechercher, ajouter et supprimer des documents.

L'interface Web est destinée à des personnes ayant de bonnes connaissances en informatique. Elle permettra de tester et visualiser le contenu de la base de donnée Gisele. Elle ne sera pas employée par des utilisateurs finaux.

Exigences fonctionnelles

Fonctionnalités

Un utilisateur devra pouvoir :

- S'authentifier.
- Consulter la liste des documents présents dans la base de donnée Gisele.
- Rechercher un document.
- Visualiser un document.
- Exporter un document.
- Supprimer un document.
- Ajouter un document.

Architecture

La plateforme devra permettre la prise en charge de nouveaux formats d'import et d'export par l'ajout de plugins.

Sécurité

Pour s'identifier, l'utilisateur devra renseigner les paramètres d'accès à la base de donnée (url, username, password). Une fois authentifié l'utilisateur aura le droit de visualiser, ajouter ou supprimer des documents.

Exigences non fonctionnelles

Logiciel

La plateforme devra être écrite en Java avec le framework JSP. Cette exigence est nécessaire pour que la plateforme soit compatible avec des composants permettant l'import/export de documents dans la base de donnée Gisele écrit en Java.

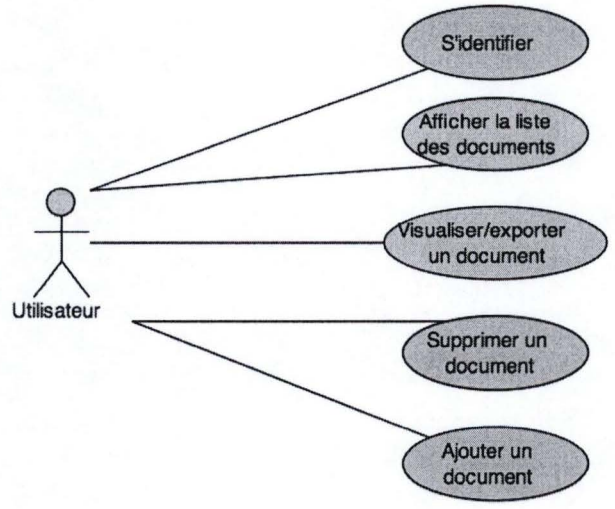
L'affichage devra être optimisé pour une résolution allant de 1024*640 à 1680*1050.

Matériel

Le serveur doit fonctionner sur une plateforme compatible avec DB-Main. Lors de l'écriture de ce rapport, les plugins DB- Main ne tournaient que sur un environnent Linux ou Windows.

Cas d'utilisation

Les cas d'utilisations de la plateforme doivent comprendre la visualisation, la recherche, l'ajout et la suppression de document ainsi que l'identification sur la plateforme.



Identification

Postcondition :

- L'utilisateur est identifié

Utilisateur	Système
1. L'utilisateur se connecte au système	
	2. Le système propose à l'utilisateur de s'identifier

3. L'utilisateur d'identifie	
	4. Le système affiche la liste des documents (UC Afficher la liste des documents)

Afficher la liste des documents

Précondition :

- L'utilisateur est identifié sur la plateforme

Utilisateur	Système
	1. Le système affiche les documents et permet à l'utilisateur d'exprimer le souhait de visualiser ou supprimer l'un d'entre eux et d'en ajouter un nouveau.
2. L'utilisateur saisit un mot clefs pour affiner la liste des documents.	
	3. Le système affiche la liste des documents contenant tout ses mots clefs.

Visualiser/exporter un document

Précondition :

- L'utilisateur est identifié sur la plateforme et se trouve au point 4 du use case identification.

Utilisateur	Système
1. L'utilisateur exprime le souhait de visualiser un document.	
	2. Le système affiche ce document de manière compréhensible (différents selon le type du document) et permet d'enregistrer le document dans

	différents format.
3. L'utilisateur choisit son format et l'endroit où il souhaite le stocker.	
	4. Le système exporte le document dans le format et à l'endroit souhaiter par l'utilisateur.

Supprimer un document

Précondition :

- L'utilisateur est identifié sur la plateforme et se trouve au point 4 du use case identification.

Utilisateur	Système
1. L'utilisateur exprime le souhait de supprimer un document.	
	2. Le système demande confirmation.
3. L'utilisateur confirme.	
	4. Le document est supprimée.

Ajouter un document

Précondition :

- L'utilisateur est identifié sur la plateforme et se trouve au point 4 du use case identification.

Utilisateur	Système
1. L'utilisateur exprime le souhait d'ajouter un document.	
	2. Le système propose un formulaire permettant de renseigner les informations nécessaires à l'ajout du document.
3. L'utilisateur complète le formulaire	
	4. Le système vérifie la validité des informations et ajoute le document si il ne détecte pas d'incohérences.

Dictionnaire des concepts

Document

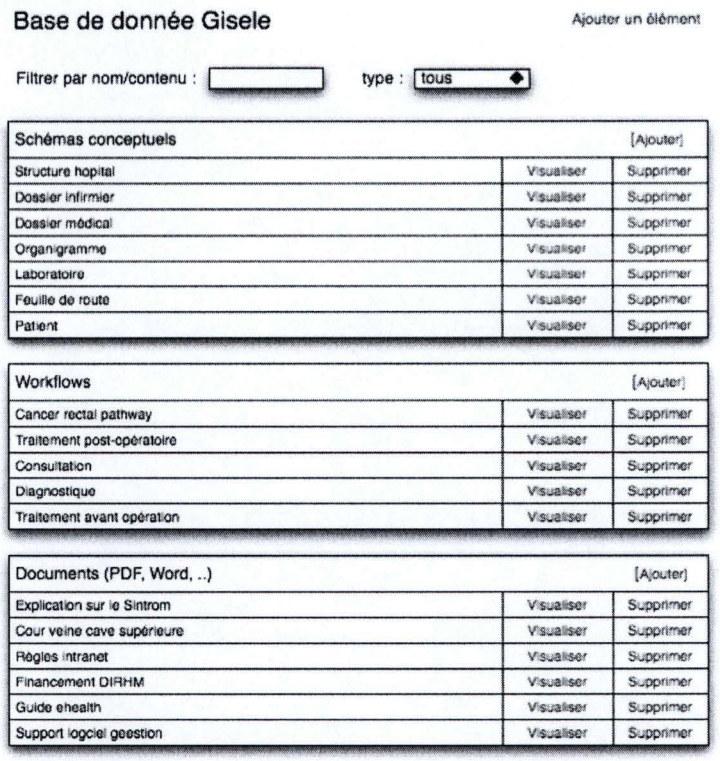
Ensembles de données se référencent entre elle et n'ayant pas de sens si elle ne sont pas vue comme un tout.

Système

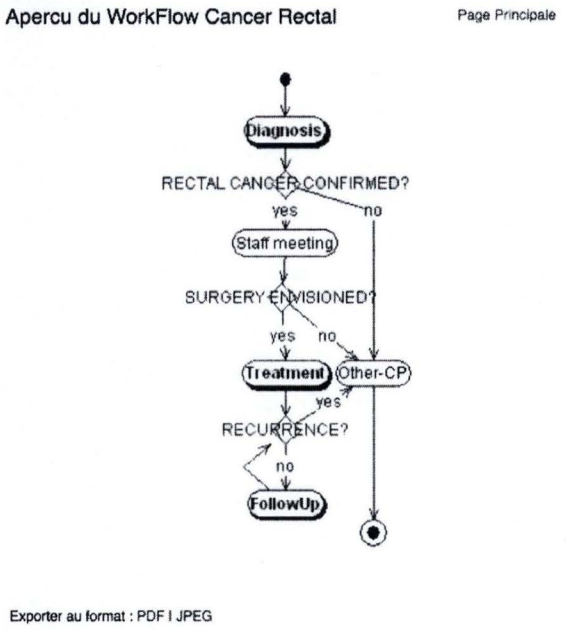
Par système, on entend le couple client/serveur qui permettent à l'utilisateur d'utiliser la plateforme.

Maquette des écrans

Page principale



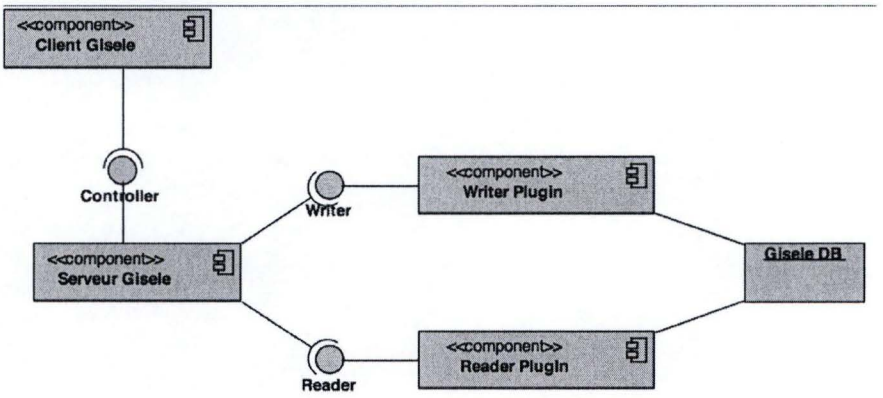
Visualisation d'un document



5.2.2. Implémentation

Architecture

L'application cliente sera structurée en un seul bloc, «ClientGisele». L'application serveur elle sera structurée en plusieurs composants : «ServeurGisele», «WriterPlugin» et «ReaderPlugin».

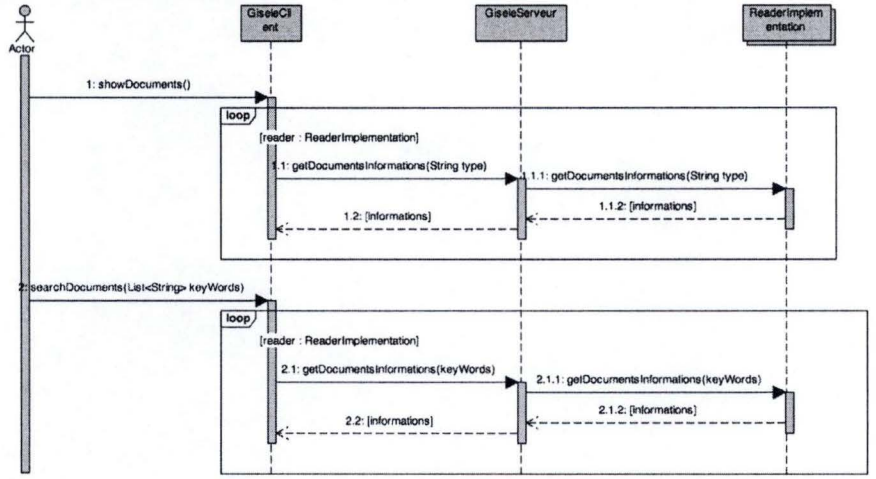


«ServeurGisele» reçoit et traite les demandes des clients qui sont de simples interfaces Web et ne contiennent pas de logique métier. Le composant «ServeurGisele» permet aussi de découvrir et communiquer avec les composants «Reader Plugin» et «Writer Plugin». Il n'y a qu'une instance de composant «GiseleServer» qui sert de contrôleur.

Les composants Reader et Writer seront chargés de créé, lire, rechercher et supprimer des documents dans la base de données Gisele. Des instances des composants de type «Reader» et «Writer» devront pouvoir être ajoutée dynamiquement.

Interactions entre composants

Affichage de la liste des documents et recherche



Affichage d'un document

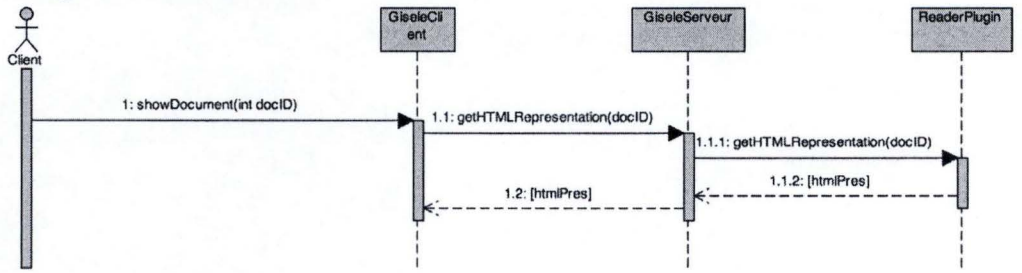
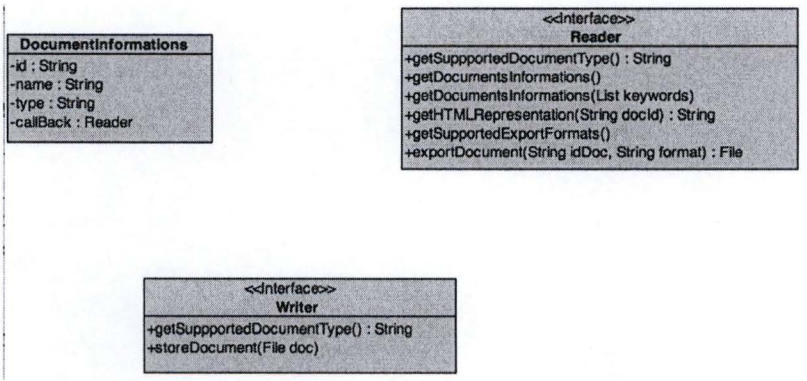


Diagramme de classe



Reader

Une classe qui implémente l'interface Reader supporte un type de document pouvant être stocké dans la base de donnée Gisele.

Un Reader doit pouvoir récupérer la liste des documents qu'il gère. Il doit permettre de représenter le document au format HTML. Si possible il doit être capable d'exporter un document dans différents formats.

Writer

Une classe qui implémente l'interface Writer permet d'enregistrer le contenu d'un fichier supporté dans la base de donnée Gisele.

DocumentInformations

Informations ou métadonnée concernant un document présent dans la base de donnée Gisele. Ces informations comprennent le nom du document, son type et une référence vers le reader capable de le lire.